Abstract

The Face Recognition Attendance System is a web-based application developed to automate the process of student attendance in a secure, efficient, and contactless manner. Traditional methods of taking attendance are often time-consuming, prone to errors, and susceptible to fraudulent entries such as proxy attendance. This project addresses these issues by integrating facial recognition technology into a web platform that utilizes real-time webcam input to identify students and mark their attendance automatically.

The system is built using the Django web framework, with facial recognition powered by facenet-pytorch, OpenCV, and Torch libraries. Initially, student face data is collected and stored during the registration phase, where each student’s facial image is pre-captured. However, only an admin has the authority to verify and approve each student’s face data for recognition purposes. Once authorized, the system can detect and recognize the student’s face in real time as they appear in front of the webcam on the website. Upon successful recognition, the system records the attendance in an SQLite3 database, which is locally hosted on the system running the application.

A key feature of the system is its interactive web-based dashboard, which displays total attendance information in a user-friendly format. This dashboard enables administrators to monitor student attendance efficiently and view daily, weekly, or total attendance statistics directly through the interface. Although the current implementation does not support data export or download functionality, it lays the foundation for future enhancements such as report generation, email notifications, and cloud storage integration.

The Face Recognition Attendance System not only reduces the administrative burden but also increases accuracy and security by eliminating manual entries and unauthorized access. It is especially useful in educational institutions where regular tracking of student attendance is crucial. With technologies such as Django, OpenCV, Torch, and SQLite3 working in harmony, this project demonstrates a practical implementation of artificial intelligence in solving real-world problems

TABLE Contents

1. Abstract ------------------------------------------------------------------------- 01
2. Table Contents ----------------------------------------------------------------- 02
3. Introduction --------------------------------------------------------------------- 03
4. Technology Stack -------------------------------------------------------------- 04
5. System Architecture / Workflow --------------------------------------------- 07
6. Block Diagram Description --------------------------------------------------- 09
7. Modules / Components -------------------------------------------------------- 11
8. Implementation Details -------------------------------------------------------- 16
9. Screenshots (With Captions) -------------------------------------------------- 18
10. Database Design --------------------------------------------------------------- 45
11. Test Cases and Outcomes ----------------------------------------------------- 47
12. Advantages & Applications --------------------------------------------------- 49
13. Challenges Faced --------------------------------------------------------------- 51
14. Future Enhancement ----------------------------------------------------------- 53
15. Conclusion ---------------------------------------------------------------------- 54
16. References ----------------------------------------------------------------------- 56
17. Appendix ------------------------------------------------------------------------ 58

**Introduction**

**Problem Statement**

In many educational institutions, the process of recording student attendance is still done manually, typically through roll calls or paper registers. This traditional method is not only time-consuming but also prone to human errors and manipulation, such as proxy attendance. As class sizes grow, maintaining accurate attendance records becomes increasingly difficult for teachers and administrators. There is a clear need for a more efficient, reliable, and automated solution that minimizes manual effort and enhances transparency in attendance tracking

**Importance of Automating Attendance**

Automating the attendance system offers numerous benefits. It saves time, reduces the workload of faculty members, and ensures accuracy by eliminating manual data entry errors. Furthermore, it helps in maintaining centralized records that can be easily accessed and monitored by authorized personnel. With the advancement of artificial intelligence and facial recognition technologies, attendance systems can now identify individuals with high precision in real time. This not only enhances security but also ensures that the process is completely contactless — a critical requirement in today’s health-conscious environment.

**Objective of the Project**

The primary objective of this project is to design and implement a web-based Face Recognition Attendance System that uses real-time webcam input to detect and recognize student faces and automatically mark their attendance. The system aims to:

* Eliminate the need for manual attendance-taking.
* Provide a secure and contactless solution.
* Offer a web interface for students and administrators to interact with the system.
* Store attendance records in a structured and retrievable format.
* Display attendance data visually through a dashboard.

**Scope of the Project**

This project is designed to work within a local network or single-system environment using a webcam for real-time face detection. It is tailored for educational institutions where students can register their facial data and get their attendance marked upon recognition. The current implementation supports:

* Student face registration (admin-authorized).
* Real-time face recognition through a web interface.
* Automatic attendance recording in a local SQLite3 database.
* A dashboard for administrators to view attendance reports.

**Technology Stack**

The successful development of the Face Recognition Attendance System relies on the integration of multiple technologies and libraries that work together to deliver a seamless, real-time, and intelligent web application. This section outlines the tools, frameworks, libraries, and programming languages used throughout the project.

**1. Backend Technology**

**Django (v5.0.7)**

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It provides built-in features for routing, authentication, database connectivity, form handling, and more. Django serves as the backbone of the application by handling:

* HTTP request/response cycle
* User authentication and session management
* Interaction with the SQLite3 database
* Admin interface for face data authorization
* Rendering of web pages using templates

**2. Frontend Technologies**

Although Django primarily handles server-side rendering, basic frontend elements are implemented using:

**HTML5, CSS3, and JavaScript**

These standard web technologies are used to build the user interface and provide structure, styling, and interactivity to the web pages.

**Jinja2 Templating Engine (v3.1.4)**

Jinja2 is used with Django to embed dynamic content into HTML templates. It allows seamless integration of data from the backend into the frontend display.

**3. Database**

**SQLite3**

SQLite is a lightweight, serverless, and self-contained relational database engine. It is integrated by default in Django and is used to store:

* Student information
* Face data
* Attendance logs  
  This database is hosted locally and ideal for single-system or prototype-level applications.

**4. Face Recognition and Deep Learning Libraries**

**facenet-pytorch (v2.6.0)**

Facenet-PyTorch is a deep learning model that provides pre-trained face recognition and face embedding functionalities. It allows the extraction of facial features, which are then used to compare and match faces during the recognition process.

**Torch (v2.2.2) & Torchvision (v0.17.2)**

These are the core libraries for tensor computation and deep learning operations. They provide the framework for running the face recognition models.

**OpenCV (v4.10.0.84)**

OpenCV (Open Source Computer Vision Library) is used for real-time image processing. In this project, it handles:

* Accessing the webcam
* Capturing video frames
* Preprocessing face images
* Drawing face bounding boxes and visual feedback

**NumPy (v1.26.4)**

NumPy supports numerical operations on image matrices and feature vectors. It is used for handling array data structures efficiently during image processing and face embedding comparisons.

**5. Supporting Python Libraries**

* **Pillow (v10.2.0):** Used for image manipulation and handling image file formats.
* **Requests (v2.32.3):** Facilitates HTTP requests for fetching or sending data, though minimally used in this project.
* **Tqdm (v4.66.4):** Adds progress bars for data processing (used during dataset preparation or training).
* **Sqlparse (v0.5.0):** Helps in parsing SQL queries internally in Django.
* **Filelock, fsspec, typing\_extensions, etc.:** These are utility libraries that support dependency management and asynchronous operations.

**6. Environment & Tools**

* **Python 3:** The core programming language used to build the application logic.
* **Virtual Environment:** Used to isolate project-specific dependencies.
* **Text Editor/IDE:** VS Code or PyCharm (recommended) for coding and debugging.
* **Browser:** Google Chrome/Firefox to run and test the web application.

**7. Deployment Environment**

* **Localhost (127.0.0.1):** The application is hosted locally using Django's development server during testing.
* **Operating System:** The project is developed and tested on Windows/Linux.

**Conclusion**

The above technology stack has been carefully selected to support the real-time, high-accuracy requirements of a face recognition system. It combines the power of Python and deep learning frameworks with Django’s robust backend capabilities and a lightweight SQLite database to deliver a complete web-based solution.

**System Architecture / Workflow**

**Overview:**

The Face Recognition Attendance System is built using a modular architecture that integrates real-time face recognition with a Django-based web application. It enables student registration, facial data training, and automatic attendance logging through webcam input. Below is the detailed workflow and architecture explanation.

**System Workflow Components**

1. **User Interface (Frontend)**
   * Built with HTML, CSS, JS, and Django templates.
   * Features:
     + **Student Registration Page** (collects name, email, reg. number, branch, year, and face image).
     + **Dashboard Page** (displays total students, attendance count, check-in/out times, filters by date).
2. **Face Registration Module**
   * Triggered when a student fills out the registration form.
   * Captures a facial image using webcam.
   * Data (form + image) is stored locally.
   * Admin authorization is required to validate the facial data for attendance recognition.
3. **Face Recognition Module**
   * Activated via real-time webcam on the home/dashboard page.
   * Uses **OpenCV** to detect face and **facenet-pytorch** + **Torch** to compare embeddings with stored images.
   * If a match is found:
     + System **marks attendance automatically**.
     + Beep sound confirms success.
     + Displays name, time, and "Check-in" or "Check-out" on screen.
     + Calculates how long the student stayed.
4. **Database Handling (SQLite3)**
   * Stores all student data, images, and attendance logs.
   * Tracks:
     + Check-in/Check-out timestamps.
     + Duration of stay.
     + Attendance records per date.
5. **Admin Dashboard**
   * Shows total registered students.
   * Displays attendance summary.
   * Allows date filtering.
   * All data is locally stored, no cloud dependency.

**Block Diagram Description**

[Student]

|

|--- Registers on Website ---> [Student Registration Page]

|

|--- Captures Form Data + Image ---> [SQLite3 DB] (pending admin auth)

[Admin]

|

|--- Verifies & Authorizes Student ---> [Trained Face Embedding Saved]

[Student Webcam Attendance]

|

|--- Student Appears on Camera ---> [Real-time Webcam Feed]

|

|--- Face Detected (OpenCV)

|--- Face Matched (Facenet, Torch)

|

|--- If Match Found ---> [Attendance Marked in SQLite3 DB]

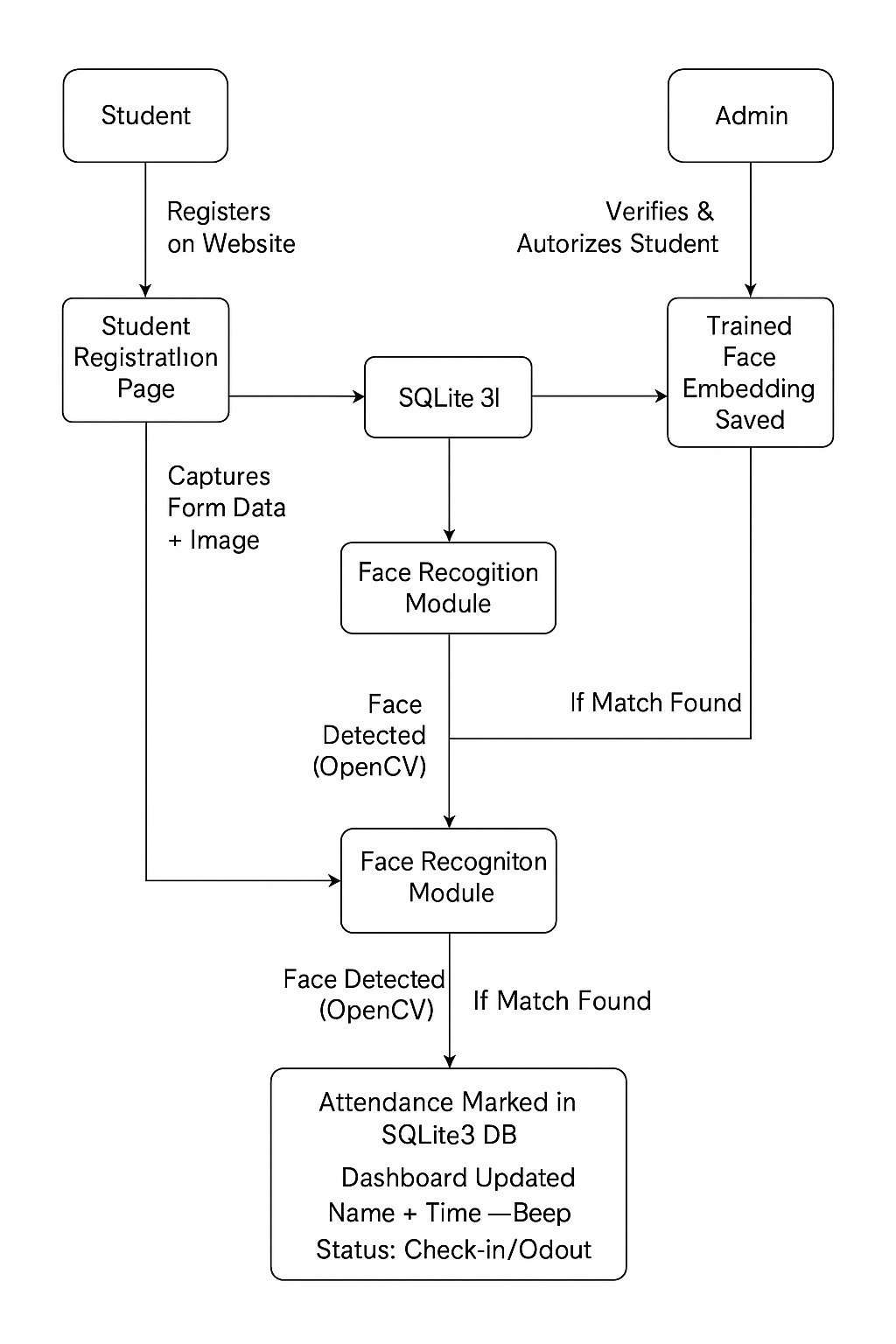
|

|--- Dashboard Updated

|--- Name + Time + Beep

|--- Status: Check-in / Check-out

|--- Duration Calculated



Modules/Components

**1. Login/Authentication Module**

The **Login/Authentication Module** is designed to restrict access to sensitive administrative functionalities of the system. Only the **admin** is allowed to log in to the system to manage student data, authorize face registrations, and view attendance reports.

* **Access Control**:  
  Only the admin has login privileges. No login or authentication is required for students to register or mark attendance.
* **Credentials**:  
  The admin must enter a valid **username** and **password** to gain access to the admin dashboard. These credentials are hardcoded or managed through Django’s built-in admin panel.
* **Authentication**:  
  The login process uses Django’s default authentication mechanism, which securely validates user credentials.
* **Session Management**:  
  The system does not include advanced session or role-based access control. Once logged in, the admin stays authenticated until logout or browser/session expiration.

This module ensures that only authorized personnel can manage attendance records and student face data, maintaining the integrity and privacy of the system.

**2. Face Detection & Recognition Module**

The **Face Detection & Recognition Module** is the core of the attendance system, responsible for identifying students in real-time using facial recognition technology.

* **Face Registration**:  
  When a student registers through the website, a **single facial image** is captured using the webcam. This image is processed and stored for facial recognition training. The captured image is later used to compare against real-time input for attendance marking.
* **Model Training and Reuse**:  
  The system does **not retrain** the model every time. Instead, the **stored image is reused**, and facial embeddings are compared using pre-trained deep learning models from **facenet-pytorch** and **Torch**.
* **Real-Time Recognition**:  
  The webcam runs continuously during the attendance session. It continuously scans for faces until the operator presses **‘q’ to quit**. This allows multiple students to be recognized automatically during a single session.
* **Face Matching**:  
  When a face is detected via the webcam (using **OpenCV**), it is compared with the stored student embeddings. If a match is found, the system proceeds to mark attendance and provides a visual and audio confirmation.

This module ensures an efficient and seamless process of identifying students and eliminates the need for manual intervention during attendance-taking.

**3. Attendance Marking Module**

The **Attendance Marking Module** is responsible for logging student attendance in real-time based on successful face recognition. It works in coordination with the Face Detection Module and database to store and manage attendance records.

* **Automatic Marking**:  
  Once a student is successfully recognized through the webcam, the system **automatically marks the attendance** without requiring any manual confirmation or input from the admin or student.
* **Check-in and Check-out Logic**:  
  The system is designed to allow **only one check-in and one check-out per student per day**. After a student has checked out, **they cannot check in again for the same day**, preventing duplicate entries and misuse.
* **Real-Time Feedback**:  
  Upon successful recognition, the system plays a **beep sound** and displays the student’s **name along with their check-in or check-out time** on the screen.
* **Duration Tracking**:  
  The module also calculates and stores how long a student stays by measuring the time between their check-in and check-out, offering additional insights into attendance behavior.

This module ensures a smooth, contactless attendance process while enforcing rules that maintain the integrity and reliability of the attendance data.

**4. Admin Panel / Attendance Report Module**

The **Admin Panel / Attendance Report Module** provides the administrator with access to student data and attendance records. While the system does not have a separate custom admin interface, Django’s built-in admin panel is leveraged for administrative tasks.

* **Access Control**:  
  There is **only one login system**, and it is restricted to the **admin**. Students do not have access to this panel. The admin logs in using a predefined **username and password**.
* **Attendance Monitoring**:  
  The admin can **view daily attendance reports** of all students through the web interface. The dashboard displays:
  + List of students with check-in/check-out times
  + Total number of students present
  + Attendance filtered by **date** using a filter option
* **Manual Data Management**:  
  While the system does not provide a graphical interface for editing entries, the **admin can manually delete records** through the **Django admin panel**. However, **editing attendance records is not allowed directly** in the web interface to maintain data authenticity.
* **Dashboard Insights**:  
  The homepage/dashboard of the website gives a summary view:
  + Total number of registered students
  + Total attendance records
  + Filter-based views by specific dates

This module plays a crucial role in managing and monitoring attendance data, ensuring that the admin has full visibility and control over the system operations.

**5. Database Integration**

The **Database Integration Module** is a critical component of the Face Recognition Attendance System, responsible for organizing and managing all data related to users, students, attendance records, and camera settings. The project uses **SQLite3**, a lightweight and file-based database that is the default backend for Django. It is ideal for this system due to its simplicity, reliability, and minimal setup.

**Database Tables**

The system primarily relies on the following four main tables:

**1. auth\_user**

* This table is part of Django’s built-in authentication system.
* It stores the credentials and metadata of users with login access, which in this case is **only the admin**.
* Fields include:
  + Username
  + Password (encrypted)
  + Email
  + Date joined
  + Last login
  + Is superuser / staff / active status

**2. app1\_student**

* This custom table handles **student registration data**.
* It is populated when a student fills out the registration form on the website.
* Fields include:
  + **Student Name**
  + **Email ID**
  + **Registration Number**
  + **Branch**
  + **Year**
  + **Authorized (Boolean)** – Indicates whether the student's face has been approved by the admin
  + **Image Path** – Stores the filename or path of the registered facial image used for recognition

**3. app1\_attendance**

* This table logs every **attendance transaction** made via face recognition.
* Each row represents a unique attendance entry for a student.
* Fields include:
  + **Student Name**
  + **Registration Number**
  + **Date of Attendance**
  + **Check-In Time**
  + **Check-Out Time**
  + **Duration of Stay**
* The system ensures **only one check-in and one check-out per day**. Once a student checks out, no further entries can be made for that day.

**4. app1\_cameraconfiguration**

* This table is used to store **camera-related settings and preferences**.
* It may include:
  + Camera source index (e.g., default webcam or external)
  + Any additional settings for resolution, frames per second, etc.

**Storage Behavior**

* **Face Images**: Captured during registration and stored locally on disk; the image path is linked to the student’s record in the database.
* **Attendance Records**: Stored locally in the database and viewable from the website’s dashboard.
* **Admin Access**: Through Django Admin Panel, the admin can **view and delete** records, but editing is restricted for data integrity.

This structured and modular database design ensures that the system maintains clean data relationships and supports reliable face-based attendance functionality.

**Implementation Details**

**Face Recognition with FaceNet-PyTorch**

This system uses the facenet-pytorch library for facial recognition. FaceNet is a deep convolutional neural network that maps face images to a compact 128-dimensional embedding space, allowing for highly accurate face comparisons.

* **Face Detection:** The MTCNN model detects faces from the webcam feed or image uploads.
* **Face Encoding:** Detected face regions are cropped, resized to 160x160 pixels, normalized, and passed through a pre-trained ResNet model to obtain a unique embedding (feature vector).
* **Face Comparison:** These embeddings are compared using Euclidean distance. If the distance between a detected face and a known face is below a threshold (e.g., 0.6), the face is considered a match.

**Camera Input with OpenCV**

OpenCV (cv2) handles real-time camera input. The system supports both:

* **Local webcams** (using integer index like 0, 1)
* **IP cameras** (using video stream URL)

Frames are read continuously until the user presses 'q' to stop. Each frame is processed for face detection and recognition.

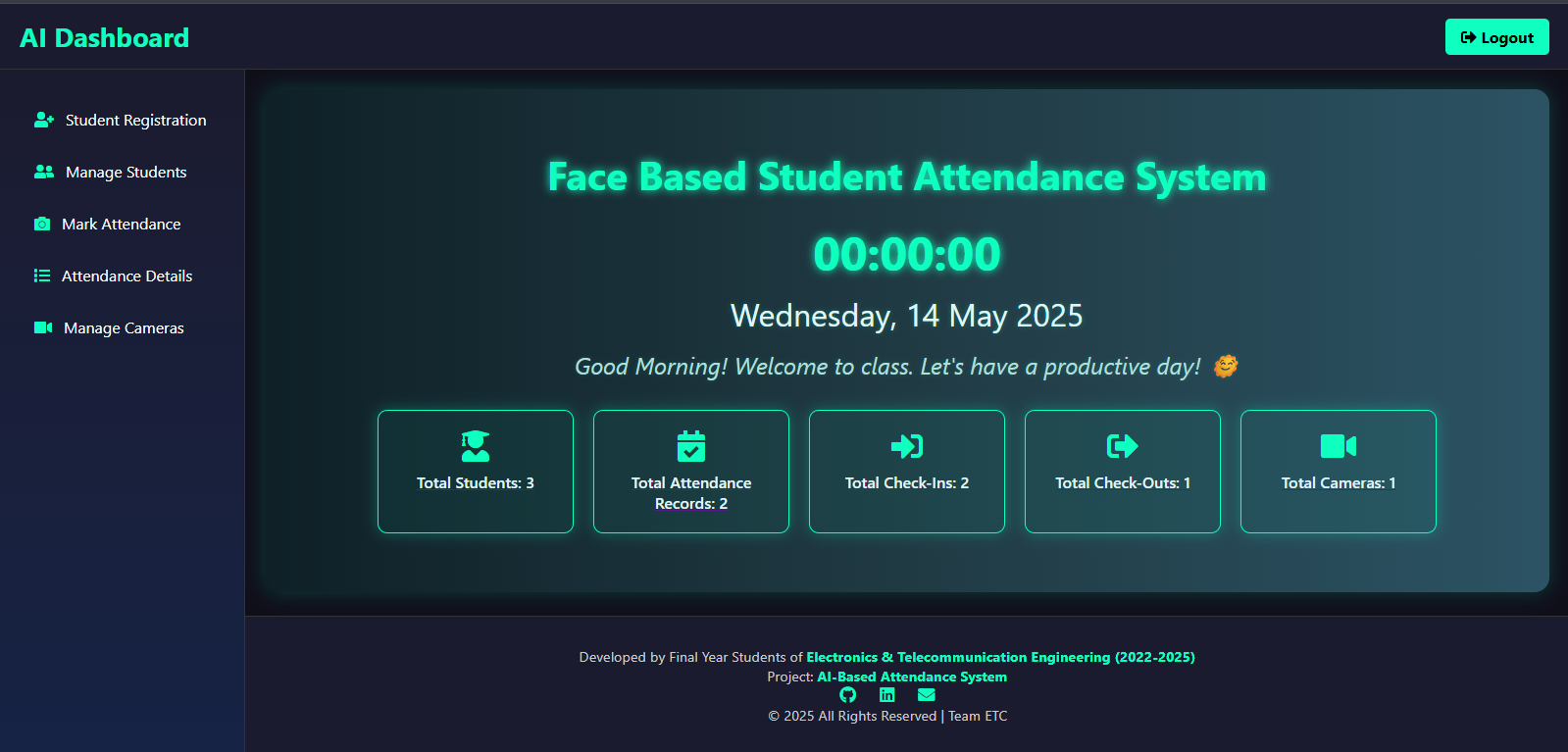
**Face Storage and Matching**

* **Storage:** During registration, a student’s face image is captured via webcam and saved in the MEDIA\_ROOT directory using Django’s file storage system.
* **Matching:** At runtime, the system loads all authorized student images, encodes them using the ResNet model, and compares them with live camera feed encodings. Recognition is performed using Euclidean distance, and attendance is marked accordingly.

**Django Routing and Page Rendering**

* **Routing:** Django URLs route requests to corresponding views. For instance:
  + /register/ → capture\_student view
  + /recognize/ → capture\_and\_recognize view
* **Templates:** HTML templates (e.g., capture\_student.html, selfie\_success.html) are rendered using Django’s template engine.
* **Forms:** Student registration forms are handled through standard POST requests. Image data is transmitted as base64 and decoded on the backend.
* **Admin Interface:** The Django Admin Panel is used by the system admin to:
  + View, delete (but not edit) attendance data
  + Authorize registered student faces

Screenshots (With Captions)



The **Home Page** of the Face-Based Student Attendance System provides a clean and modern dashboard view for users. The UI is designed using a dark theme with glowing neon highlights for better readability and aesthetics. Here's a detailed breakdown:

**Navigation Sidebar (Left Panel)**

The sidebar provides quick access to the following core functionalities:

* **Student Registration** – To register new students with facial data.
* **Manage Students** – View or manage existing student data.
* **Mark Attendance** – Start real-time face recognition and mark attendance.
* **Attendance Details** – View attendance logs, check-ins, and check-outs.
* **Manage Cameras** – Add or manage camera sources.

Each menu item is accompanied by an intuitive icon for ease of use.

**Dashboard Section (Main Panel)**

* **System Title**: *"Face Based Student Attendance System"* is displayed prominently.
* **Live Clock and Date**: Real-time digital clock and the current date are shown for context.
* **Welcome Message**: A motivational greeting ("Good Morning! Welcome to class...") adds a friendly touch.

**Summary Cards (Metrics)**

Five cards show real-time statistics:

1. **Total Students**: Number of registered students (e.g., 3).
2. **Total Attendance Records**: Cumulative attendance entries (e.g., 2).
3. **Total Check-Ins**: Students currently marked as present (e.g., 2).
4. **Total Check-Outs**: Students marked as left (e.g., 1).
5. **Total Cameras**: Number of active camera devices configured (e.g., 1).

Each card features a relevant icon and neon-glow effect for visibility.

**Footer**

The footer gives project credits:

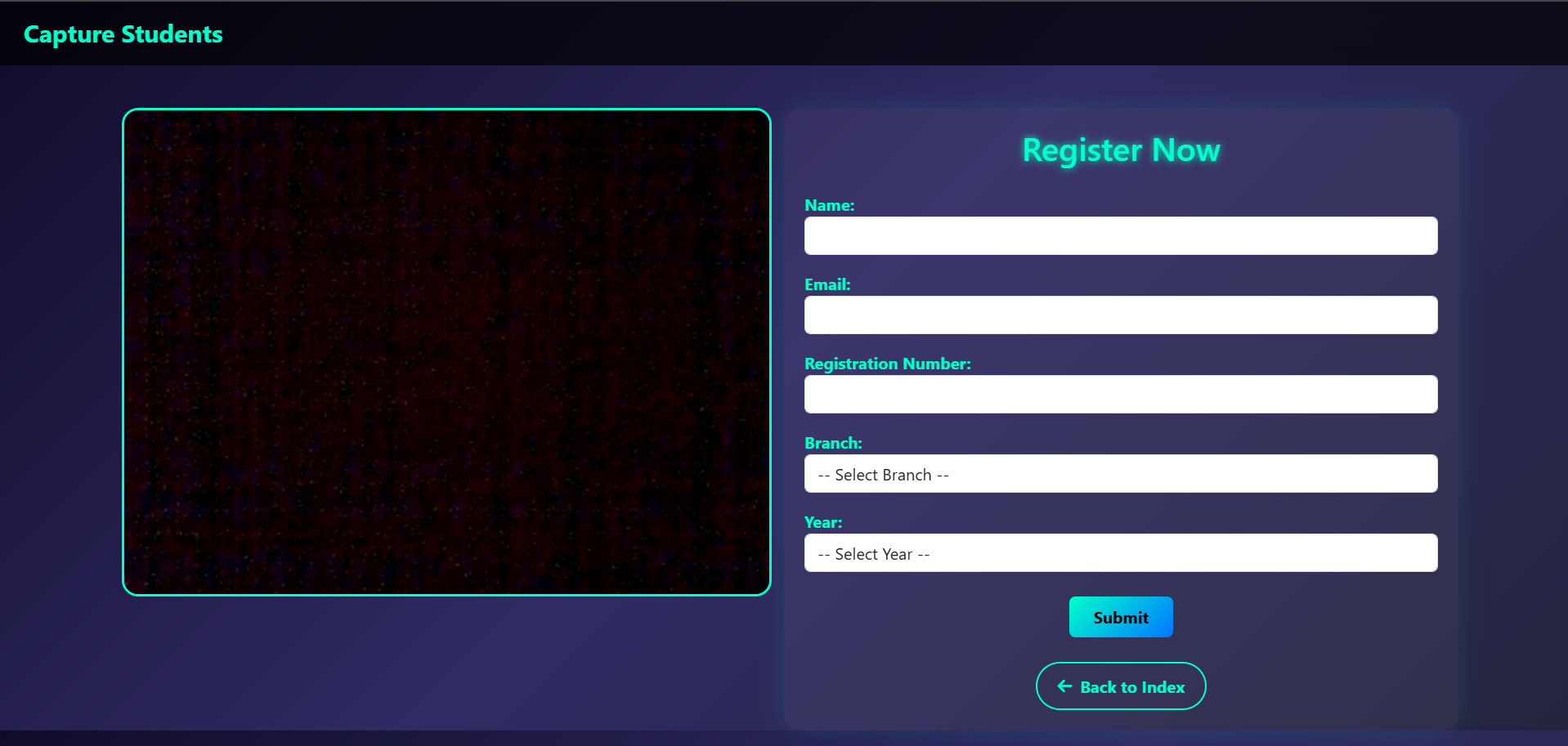
* Developed by **Final Year Students of Electronics & Telecommunication Engineering (2022–2025)**.
* Project Title: **AI-Based Attendance System**
* Social media icons (LinkedIn, GitHub) indicate possible links to team profiles or repositories.
* Branding: “© 2025 All Rights Reserved | Team ETC”

**Login and Logout Functionality**

The system includes secure session handling via a **Login** and **Logout** mechanism to ensure only authorized users (such as administrators or teachers) can access the attendance dashboard.

* **Login Page**:
  + When the application is **logged out**, the system redirects to a **Login Page**.
  + Users are required to enter valid credentials (e.g., username/email and password) to gain access.
  + Upon successful login, users are taken directly to the **AI Dashboard** (Home Page).
* **Logout Button**:
  + Located at the top-right corner of the dashboard.
  + On clicking **Logout**, the session is terminated and the user is redirected back to the Login Page.
  + This ensures that student data and attendance records remain protected.

This authentication flow is crucial for maintaining the privacy and security of sensitive student information.



**Student Registration Page**

The **Student Registration Page** is a crucial component of the Face-Based Attendance System. It allows new students to register into the system by entering their details and capturing their facial data, which is later used for automated attendance marking.

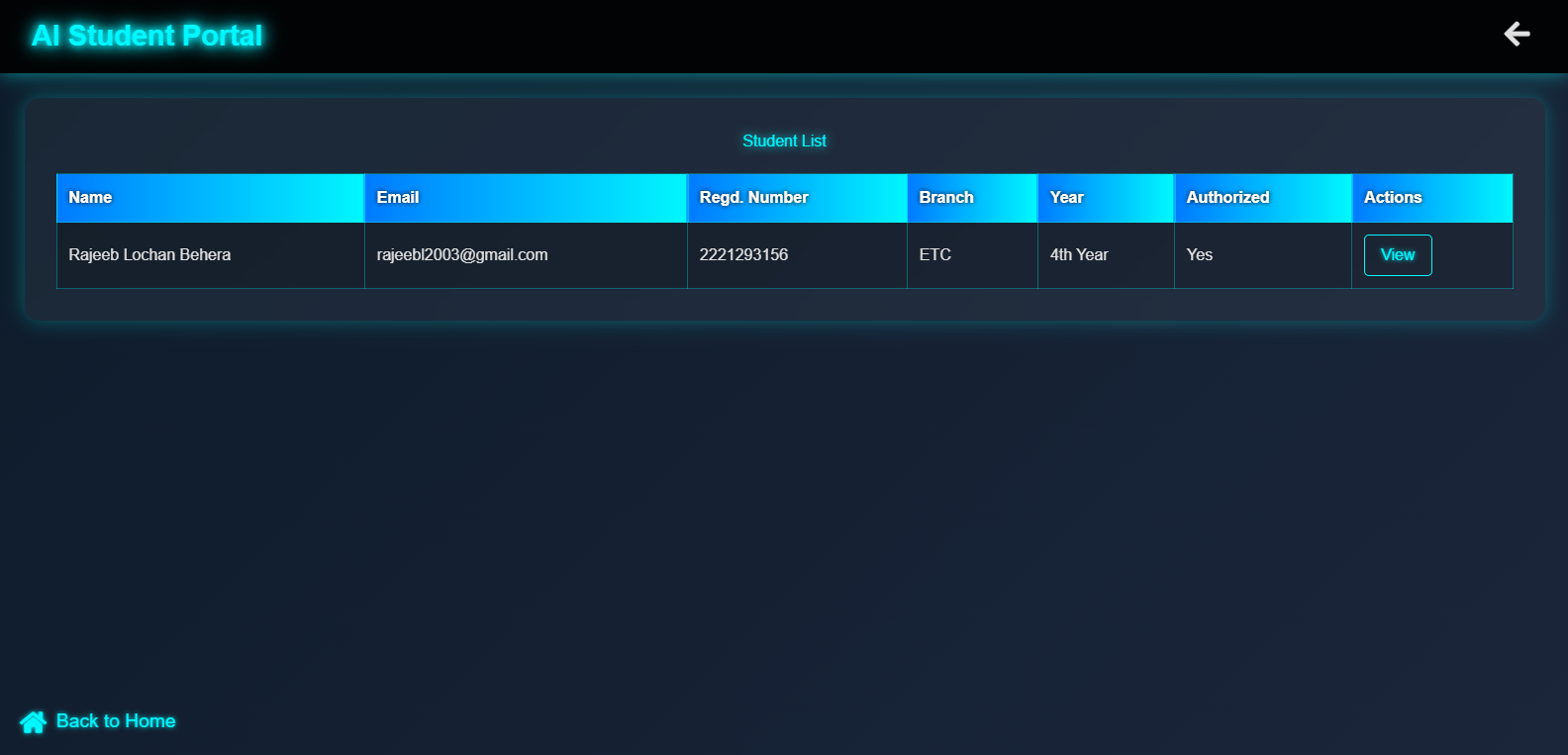
**Key Features:**

* **Live Camera Feed:**  
  The left section displays a real-time camera feed for capturing the student’s face. This feed is essential for collecting accurate facial data used in recognition and attendance tracking.
* **Registration Form:**  
  On the right side, students are required to fill out a registration form that includes:
  + **Name**
  + **Email**
  + **Registration Number**
  + **Branch** (selectable from a dropdown)
  + **Year** (selectable from a dropdown)
* **Buttons:**
  + **Submit:** Saves the entered data and captured facial images to the system database for future recognition.
  + **Back to Index:** Navigates back to the main index or home page.

**Functionality:**

Once the student completes the form and clicks **Submit**, the system:

1. Captures multiple images from the live feed.
2. Stores these images in the dataset directory with the associated student details.
3. Prepares the data for the facial recognition training model.



**Total Student Page (Student List View)**

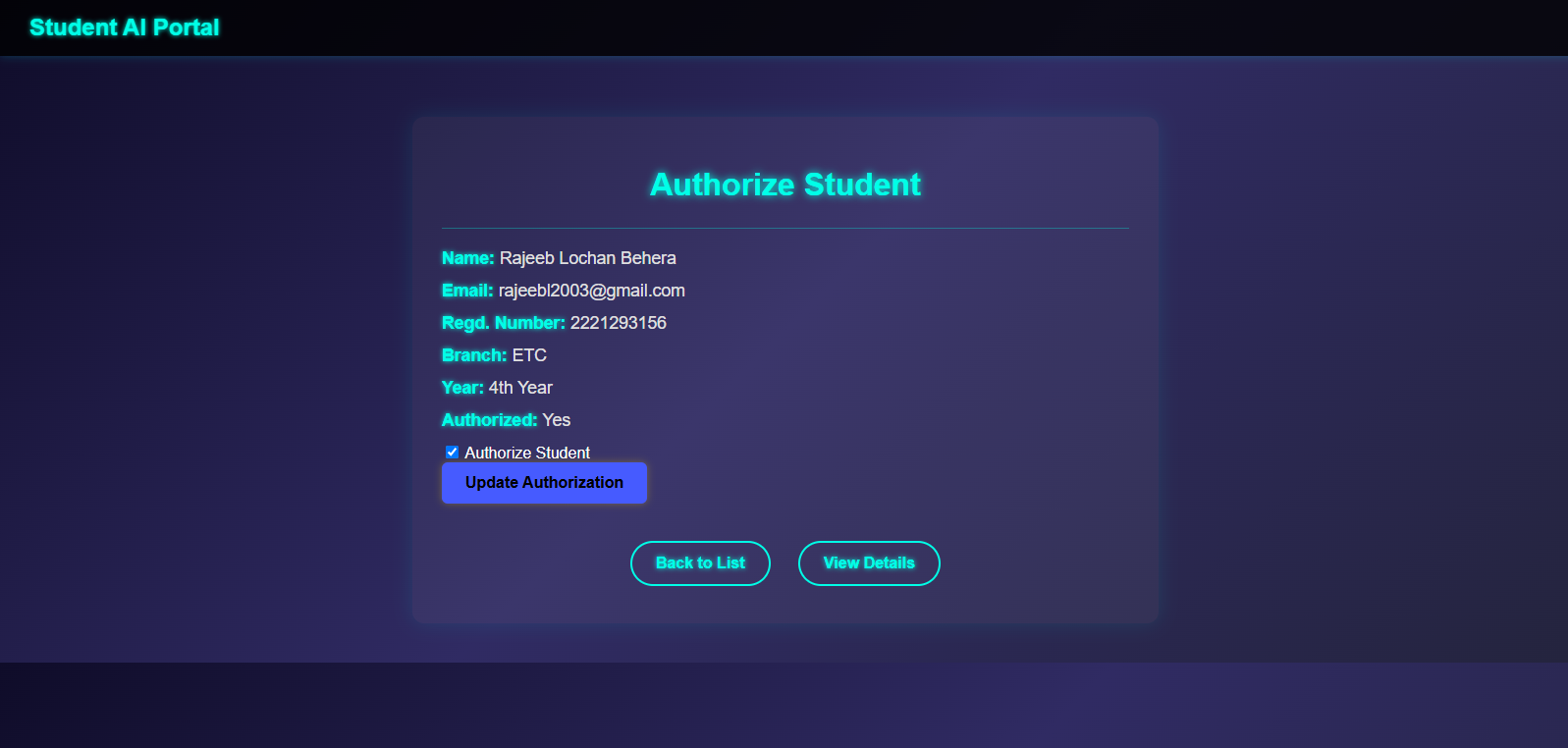
This section displays a list of all registered students in the **AI-Based Student Attendance System**. It provides a quick overview of essential student details in a tabular format.

**Key Features:**

* **Title**: The top of the page is labeled **AI Student Portal**, with a subtitle **Student List** to indicate the current view.
* **Displayed Fields**:
  + **Name**: Full name of the student.
  + **Email**: Registered email address.
  + **Regd. Number**: Unique registration number.
  + **Branch**: Department or stream (e.g., ETC).
  + **Year**: Academic year (e.g., 4th Year).
  + **Authorized**: Indicates if the student is authorized (Yes/No).
  + **Actions**: Contains a **View** button to access detailed student information.

**Navigation**:

* A **Back to Home** link is available at the bottom-left for easy navigation to the dashboard.
* A back arrow icon in the top-right corner offers additional navigation functionality.



**Authorize Student Page**

This page is displayed when the admin clicks the **"View"** button from the Total Student List. It allows the admin to review and manage a student’s authorization status.

**Key Features:**

* **Title**: The top of the section is labeled **Authorize Student**.
* **Student Information Displayed**:
  + **Name**: Student’s full name.
  + **Email**: Student's email ID.
  + **Regd. Number**: Registration number.
  + **Branch**: Department or branch of study.
  + **Year**: Academic year.
  + **Authorized**: Current authorization status (Yes/No).
* **Admin Controls**:
  + A **checkbox** labeled “Authorize Student” allows the admin to toggle authorization status.
  + **Update Authorization** button saves the changes to the student’s status in the system.
* **Navigation Buttons**:
  + **Back to List**: Redirects the admin back to the Total Student List page.
  + **View Details**: Redirects to a detailed student information page .

**Purpose**:  
This page provides an intuitive interface for the administrator to authorize or revoke access for students to use the attendance system (e.g., face recognition system).



**Total Attendance Report Page**

This page allows admins to view and manage attendance logs of all students captured by the face recognition system.

**Header Title**:

* **Student Attendance Records** – clearly indicates the purpose of the page.

**Search & Filter Options**:

* **Search Bar**: Admins can search attendance records by student name or registration number.
* **Date Filter**: Allows filtering of attendance records by a specific date using a date picker.
* **Search/Filter Buttons**: Used to execute search and filter queries for quick access to specific records.

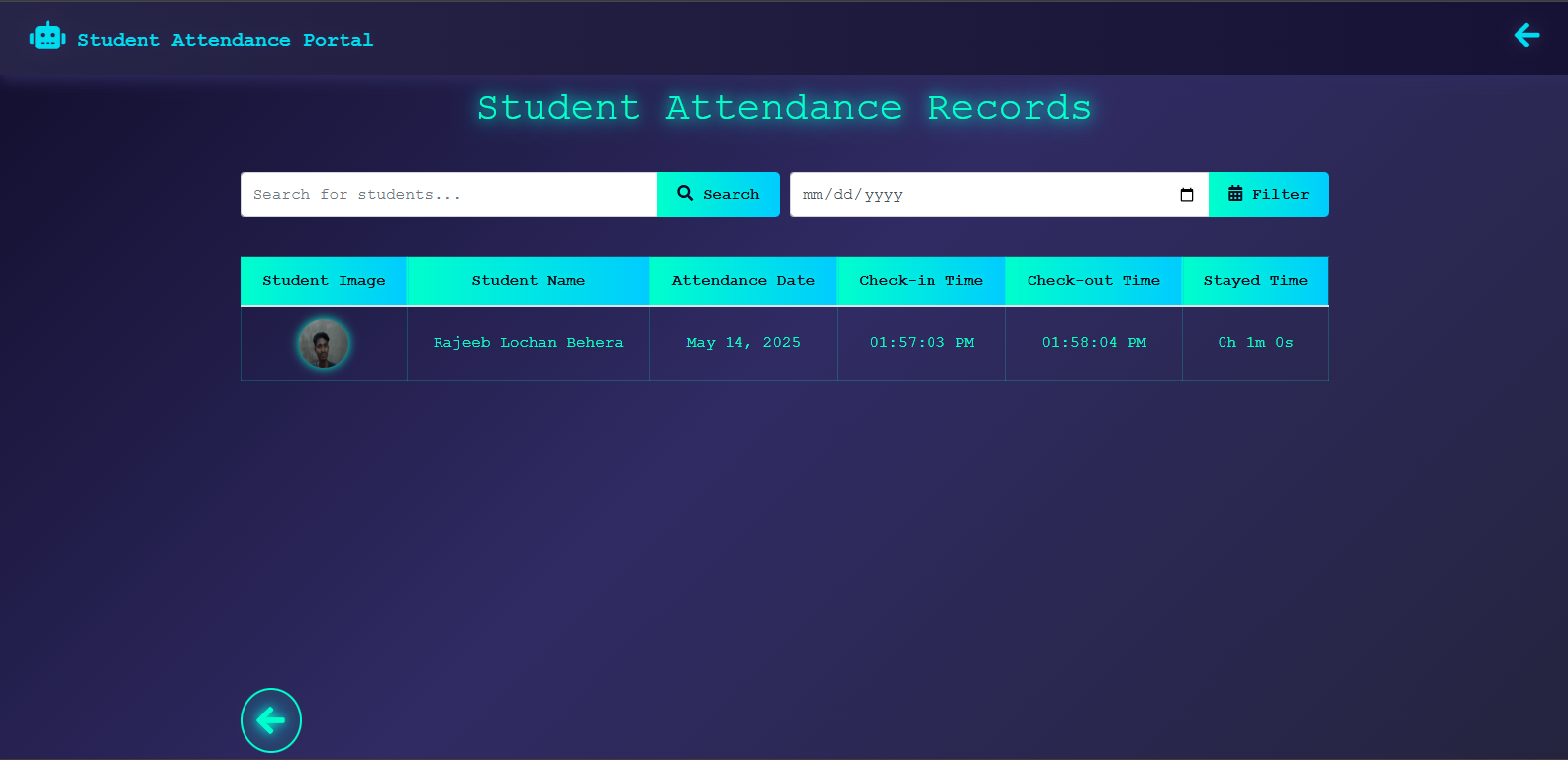
**Attendance Table Columns**:

* **Student Image**: Displays the captured image of the student at the time of attendance marking.
* **Student Name**: Full name of the student.
* **Attendance Date**: The date on which attendance was recorded.
* **Check-in Time**: Time the student was recognized and marked present.
* **Check-out Time**: Time the student left and system recorded their exit.
* **Stayed Time**: Duration for which the student was present (calculated as difference between check-in and check-out time).

**Navigation**:

* A **Back Button** (bottom left corner) allows navigation to the previous admin dashboard or list page.

**Purpose**:  
This page serves as a centralized log for monitoring and verifying student attendance, providing visual confirmation and exact timestamps. It enhances transparency and traceability in attendance management.



**Total Attendance Report Page**

This page allows admins to view and manage attendance logs of all students captured by the face recognition system.

**Header Title**:

* **Student Attendance Records** – clearly indicates the purpose of the page.

**Search & Filter Options**:

* **Search Bar**: Admins can search attendance records by student name or registration number.
* **Date Filter**: Allows filtering of attendance records by a specific date using a date picker.
* **Search/Filter Buttons**: Used to execute search and filter queries for quick access to specific records.

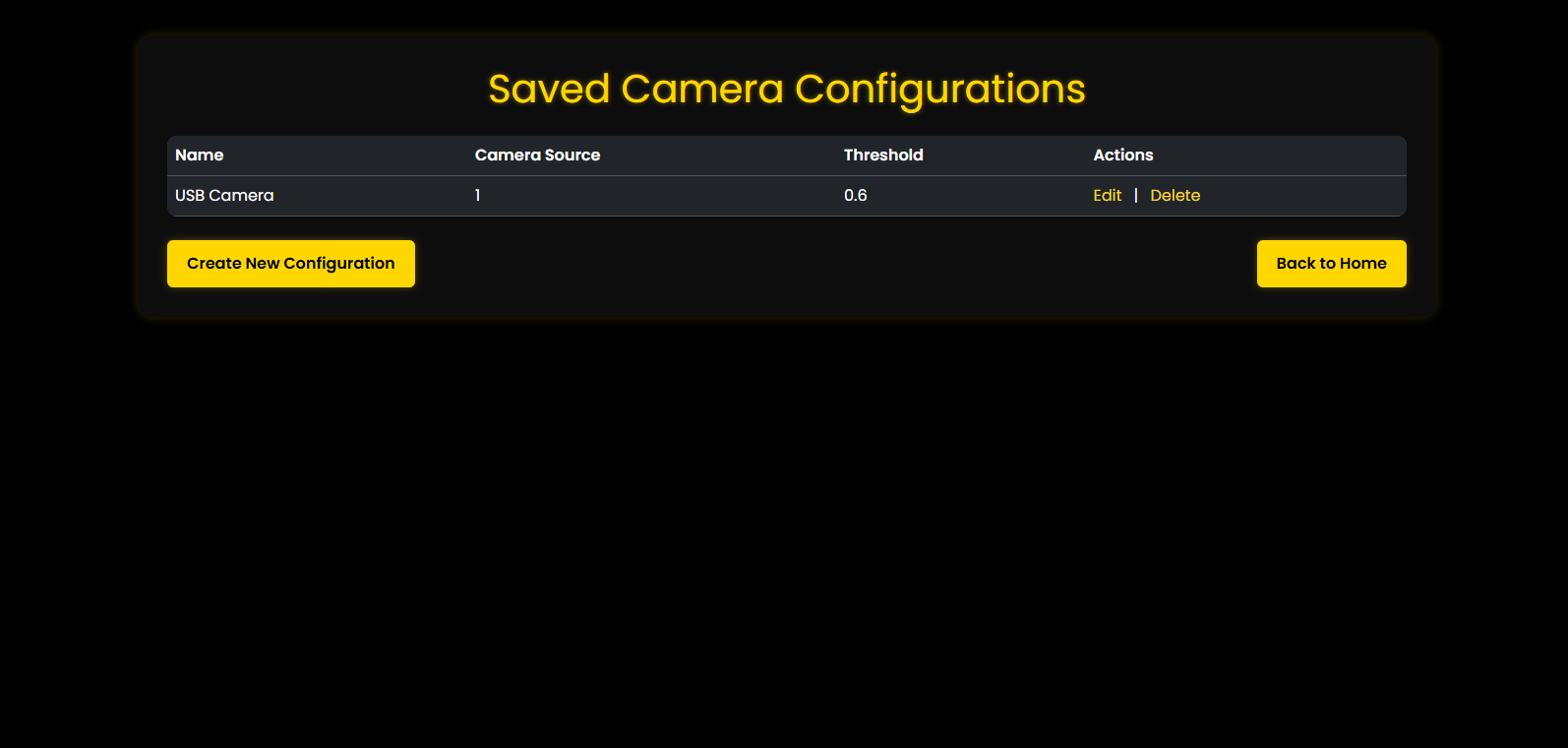
**Attendance Table Columns**:

* **Student Image**: Displays the captured image of the student at the time of attendance marking.
* **Student Name**: Full name of the student.
* **Attendance Date**: The date on which attendance was recorded.
* **Check-in Time**: Time the student was recognized and marked present.
* **Check-out Time**: Time the student left and system recorded their exit.
* **Stayed Time**: Duration for which the student was present (calculated as difference between check-in and check-out time).

**Navigation**:

* A **Back Button** (bottom left corner) allows navigation to the previous admin dashboard or list page.

**Purpose**:  
This page serves as a centralized log for monitoring and verifying student attendance, providing visual confirmation and exact timestamps. It enhances transparency and traceability in attendance management.



**Camera Configurations**

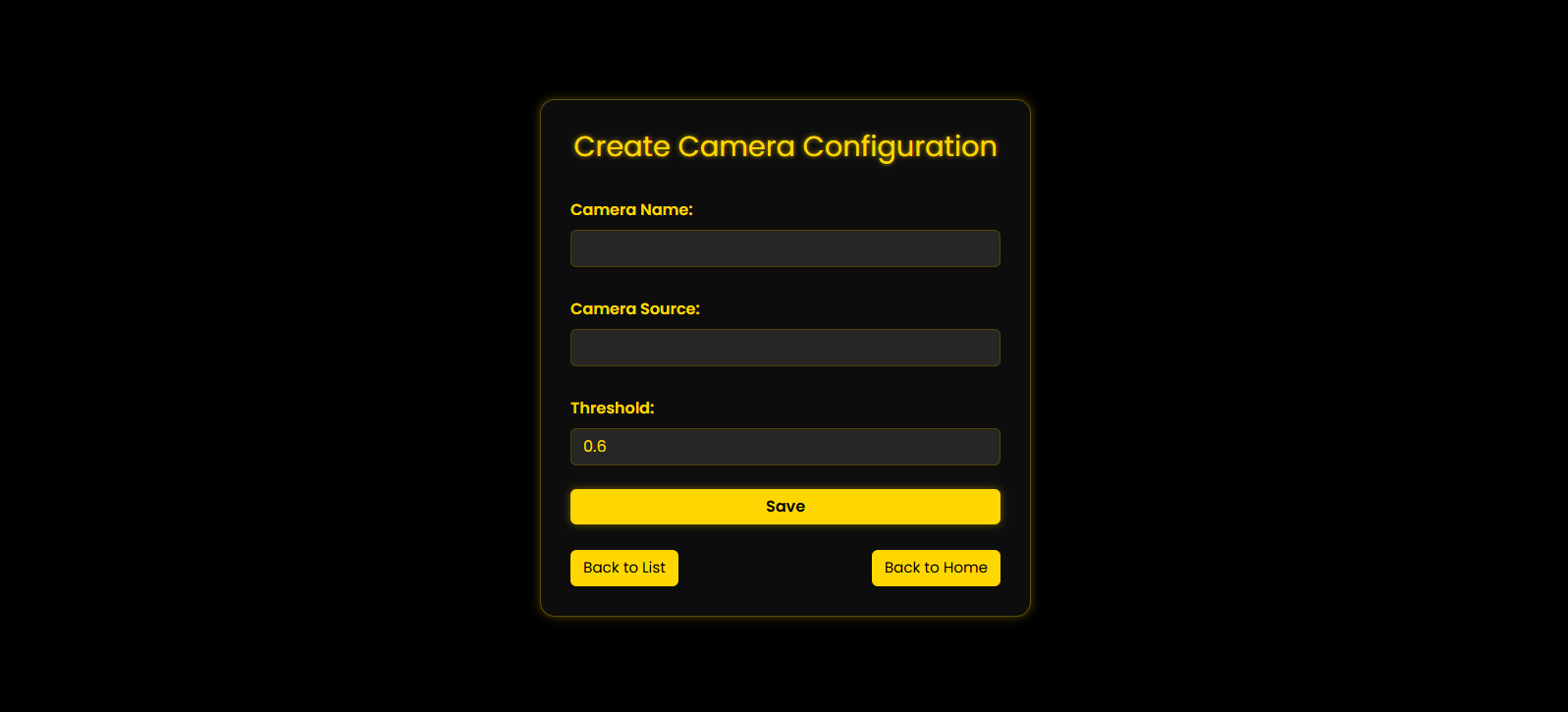
**Purpose**

This page allows administrators or users to view, manage, and configure camera sources used for face recognition in an attendance system. Each configuration defines parameters required for the system to interact with a specific camera, likely used for capturing and recognizing faces in real-time.

1. **Header Title**
   * Text: **"Saved Camera Configurations"**
   * Style: Prominently displayed with bold, yellow text to indicate the page’s primary function.
2. **Configuration Table**
   * **Columns:**
     + **Name**: Descriptive label for the camera (e.g., “USB Camera”).
     + **Camera Source**: Numerical identifier or index representing the camera device (e.g., 1 might correspond to /dev/video1).
     + **Threshold**: A numerical value (e.g., 0.6) likely representing the confidence threshold for face recognition.
     + **Actions**: Options to **Edit** or **Delete** the existing configuration.
3. **Action Buttons**
   * **Create New Configuration**: Button styled in yellow, allowing users to add a new camera setup.
   * **Back to Home**: Navigation button to return to the main dashboard or homepage of the application.

**Functional Description**

* The page presents a **list of configured camera sources** with associated thresholds that determine how strictly the system should match detected faces.
* Each row in the configuration table represents a **distinct camera setup** which can be modified or removed using action links:
  + **Edit**: Opens a form or modal to update the camera name, source index, or recognition threshold.
  + **Delete**: Removes the configuration, possibly with a confirmation prompt to avoid accidental deletion.
* The **“Create New Configuration”** button likely navigates to a new form or opens a modal where users can define a new camera source, assign a label, and set the threshold.
* **Back to Home** simplifies navigation for administrators, suggesting this page is part of a larger **admin dashboard or portal**.



**Camera Configuration**

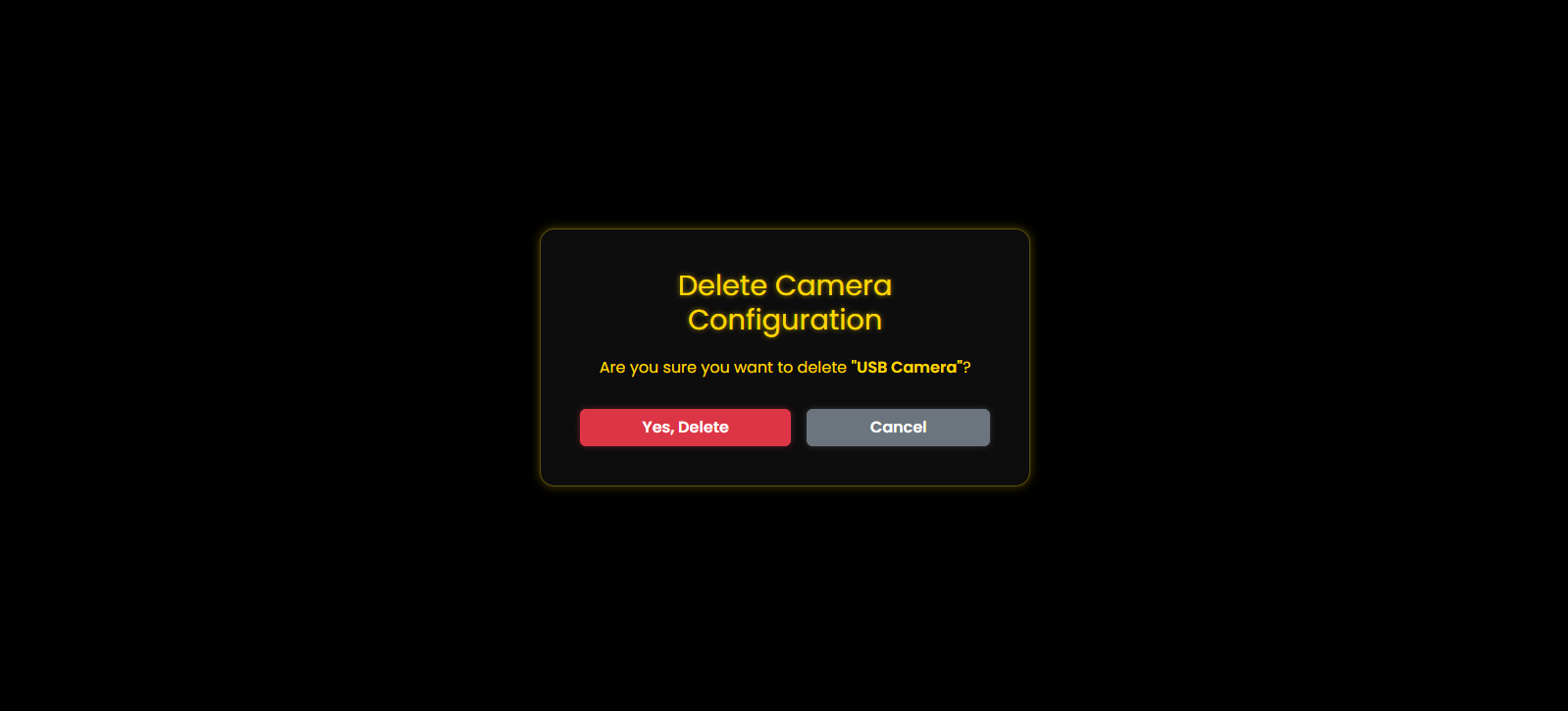
**Purpose**

This page allows administrators to **add a new camera setup** to the system. Each camera configuration includes a name, source identifier, and a confidence threshold for face recognition. This is essential for enabling or expanding real-time face capture functionality.

1. **Header Title**
   * **Text**: “Create Camera Configuration”
   * **Style**: Bold, glowing yellow text that visually emphasizes the form’s purpose.
2. **Input Fields**
   * **Camera Name**:
     + Placeholder or Label: "Enter Camera Name"
     + Purpose: A descriptive name for the camera (e.g., "Main Gate Cam", "Lab Cam").
   * **Camera Source**:
     + Placeholder or Label: "Enter Camera Input ID"
     + Purpose: Numerical or string identifier used by the backend to access the camera device (e.g., 0, 1, or /dev/video0).
   * **Threshold**:
     + Default Value: 0.6
     + Purpose: A float value that defines the **minimum face match confidence**. The system will only consider matches above this threshold valid.
3. **Action Buttons**
   * **Save**:
     + Prominent yellow button to submit the form.
     + Function: Triggers the addition of the new configuration to the system/database.
   * **Back to List**:
     + Returns the user to the camera configurations list view (previous screen).
   * **Back to Home**:
     + Navigates back to the main dashboard or homepage of the admin portal.

**Functional Description**

* **Form-Driven Input**: The user is expected to manually enter all three values—camera name, source ID, and threshold.
* **Threshold Field**: Pre-filled with a common value (0.6) to guide users toward a suitable default, but still editable.
* **On Submit**: Likely calls a backend route or controller to persist the new configuration in a database.
* **Navigation Controls**:
  + **Back to List** ensures quick return to view/manage other configurations.
  + **Back to Home** supports broader navigation within the system

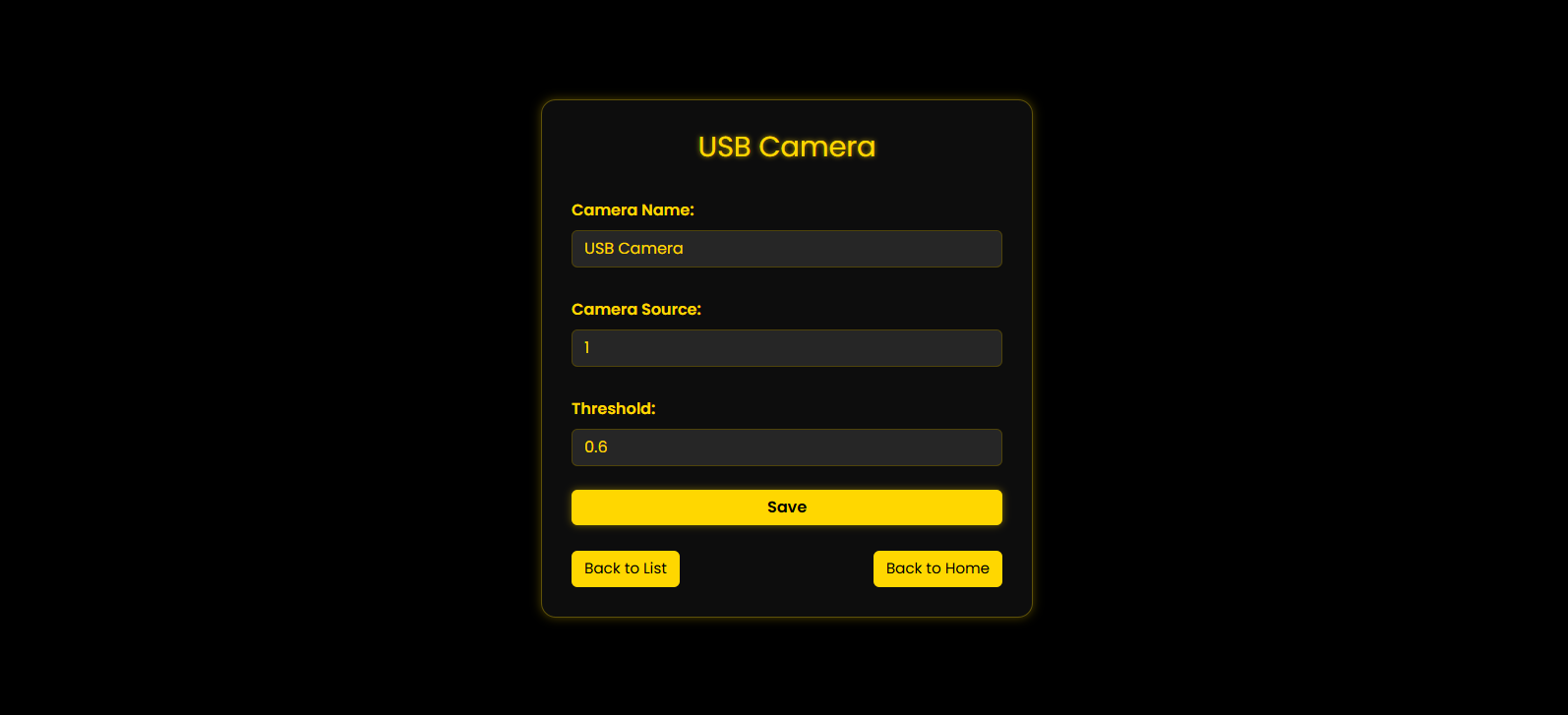


**Delete Camera Configuration - Confirmation Modal**

**Purpose**

This modal is used to **confirm the deletion** of a camera configuration, preventing accidental removals.

1. **Title:**
   * Text: **"Delete Camera Configuration"**
   * Style: Bold, centered, glowing yellow to match system branding.
2. **Message:**
   * **Confirmation Prompt**:  
     Are you sure you want to delete "USB Camera"?
   * The camera name (USB Camera) is dynamically inserted based on user selection.
3. **Action Buttons:**
   * **Yes, Delete (Red Button)**:
     + Executes the deletion.
     + Color indicates a destructive action (standard UX pattern).
   * **Cancel (Gray Button)**:
     + Dismisses the modal with no change made.



**Edit Camera Configuration – USB Camera**

**Purpose**

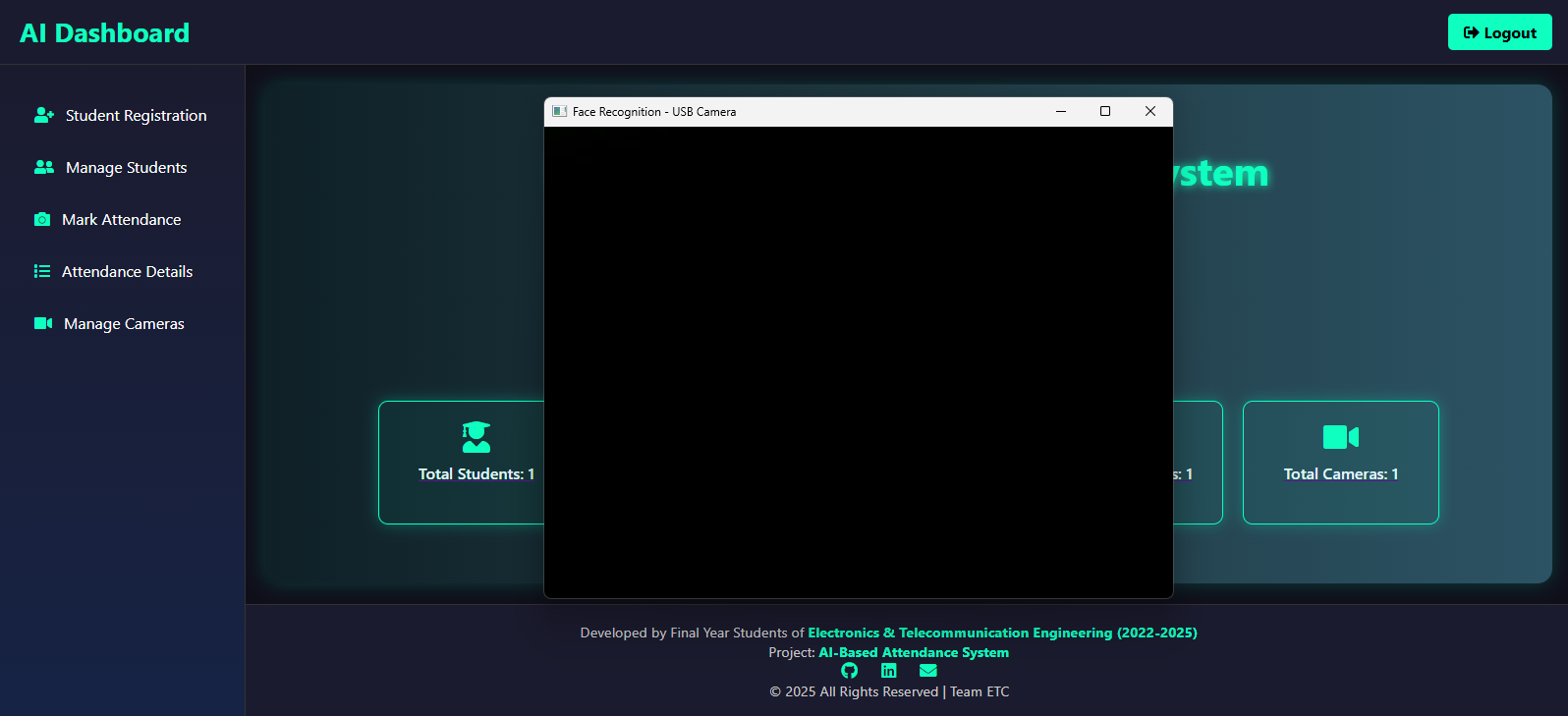
This page allows the user to **modify settings** for a selected camera used in the face recognition attendance system.

**UI Components Overview**

| **Field** | **Description** |
| --- | --- |
| **Camera Name** | Text input for naming/renaming the camera (e.g., *USB Camera*) |
| **Camera Source** | Numeric ID or path used by the system to access the camera (e.g., 0, 1, or a video stream URL) |
| **Threshold** | Float value that defines the **confidence threshold** for face recognition (e.g., 0.6) |

**Action Buttons**

* **Save** (yellow):
  + Saves the changes made to the camera configuration.
  + Likely triggers an API call or updates local state/database.
* **Back to List**:
  + Navigates to the camera configuration list page.
* **Back to Home**:
  + Returns to the application’s main dashboard or homepage.



**Mark Attendance (Main Dashboard Area Center Panel)**

**Live Camera Feed Window**

* **Title**: "Face Recognition – USB Camera"
* Currently shows a black screen, possibly indicating:
  + The camera is initializing,
  + No face is detected,
  + Or the camera is not functioning properly.
* This pop-up is essential for the **Mark Attendance** function, where students’ faces are captured and matched against stored records.

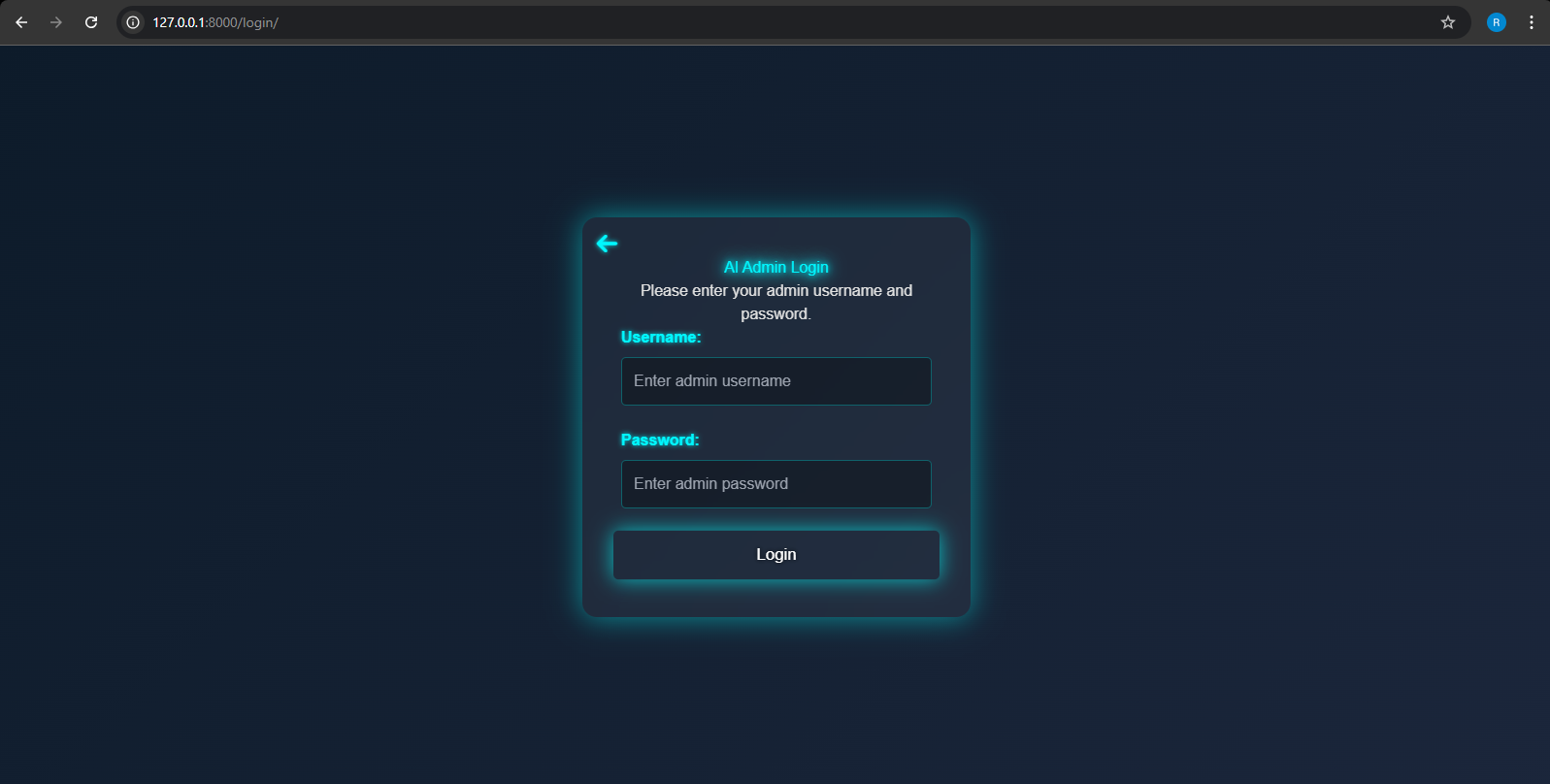
**Functional Description**

This page serves as the central hub for managing the AI-powered attendance system. Specifically, the **Mark Attendance** module is activated here, opening a camera interface to recognize and verify student faces.

Here’s how it works:

1. The admin clicks "Mark Attendance."
2. A new window (seen in the image) opens to access the USB camera.
3. The system uses real-time face recognition to identify registered students.

If a match is found, attendance is marked automatically in the database.



**Admin Login**

**Purpose**

This page serves as the **authentication gateway** for accessing the admin dashboard of the AI-Based Attendance System. Only authorized users (admins) can log in and manage the system.

**1. Page Layout**

* **Centered Login Box**: The login form is displayed in the center with a glowing neon border effect, maintaining a consistent futuristic theme with the main dashboard.
* **Background**: A dark blue gradient or flat background adds to the tech-centric design and contrasts well with the input elements.

**2. Form Elements**

* **Username Field**:
  + Label: "Username:"
  + Placeholder: "Enter admin username"
* **Password Field**:
  + Label: "Password:"
  + Placeholder: "Enter admin password"
  + The password is masked as it's typed.
* **Login Button**:
  + Label: "Login"
  + Likely triggers backend verification and session initialization.
  + Styled with a glowing border to match the theme.

**3. Title & Instructions**

* **Header**: *AI Admin Login* (highlighted in neon cyan).
* **Instruction Text**: "Please enter your admin username and password."

**4. Navigation**

* **Back Icon (Top Left)**: Indicates the option to navigate back to a previous page or landing screen. Likely useful if this login screen is not the default homepage.

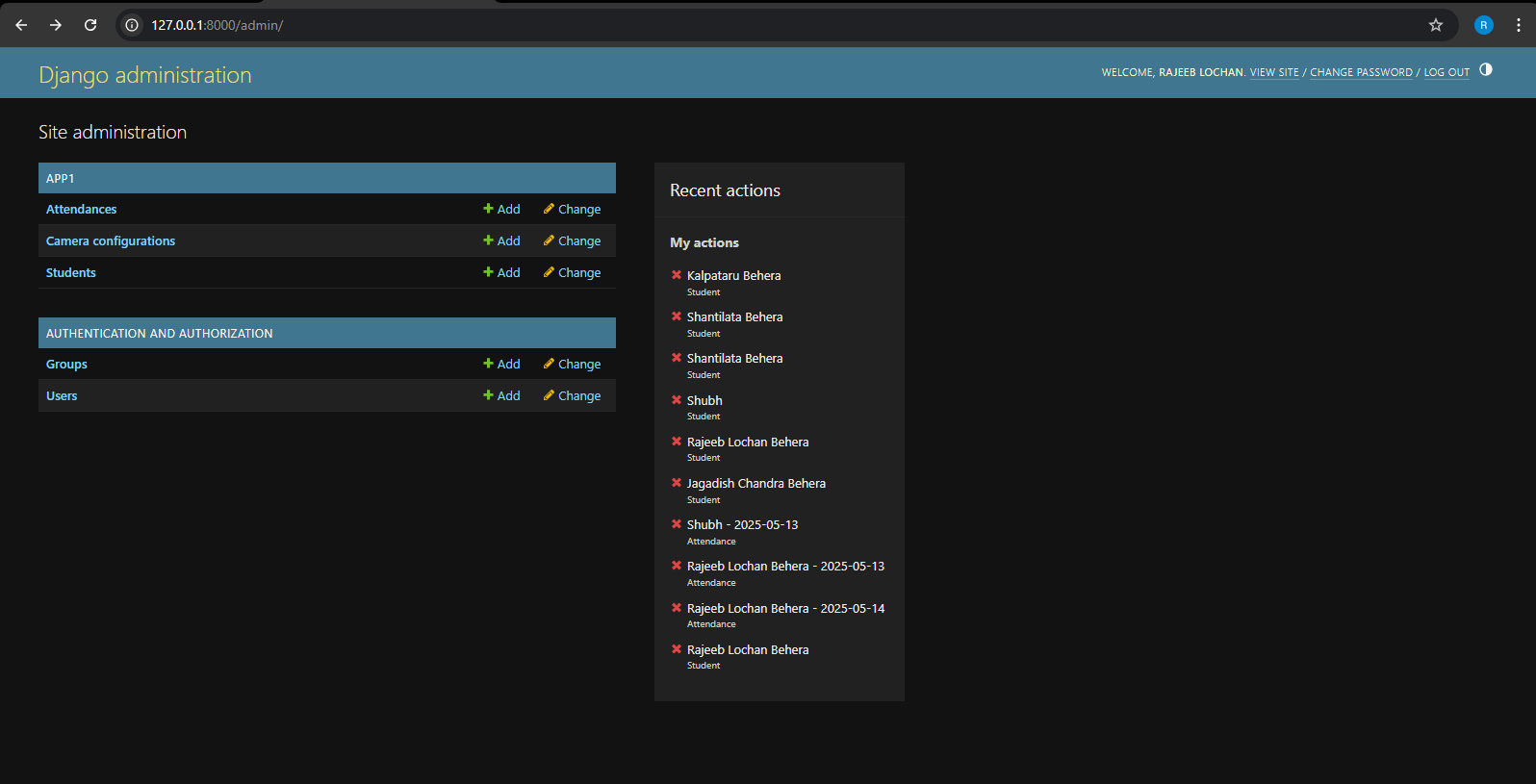
**Functional Description**

When an admin enters their credentials:

1. The **form is submitted**, typically via a POST request to the backend.
2. The system checks the **authentication details**.
3. If verified successfully, the user is **redirected to the Home Dashboard**, as seen in the first screenshot you shared.
4. On failed login, the system likely shows an error (not shown here but usually part of such systems).

This login page ensures **secure access** to the administrative tools for:

* Student management,
* Attendance marking via face recognition,
* System camera configuration.



**Django Admin Panel**

**Purpose**

This admin panel is the **backend administrative interface** generated automatically by Django. It allows superusers or staff to manage database models directly (e.g., attendances, students, and camera configurations) through a user-friendly interface.

**1. Header Section**

* **Title:** Django administration
* **User Info:** Shows the currently logged-in admin — “RAJEEB LOCHAN”.
* **Links:**
  + VIEW SITE: Takes you to the public-facing site (if configured).
  + CHANGE PASSWORD: Allows the admin to update their login credentials.
  + LOG OUT: Ends the admin session.

**2. Main Admin Menu (Left Panel)**

**App1 Section**

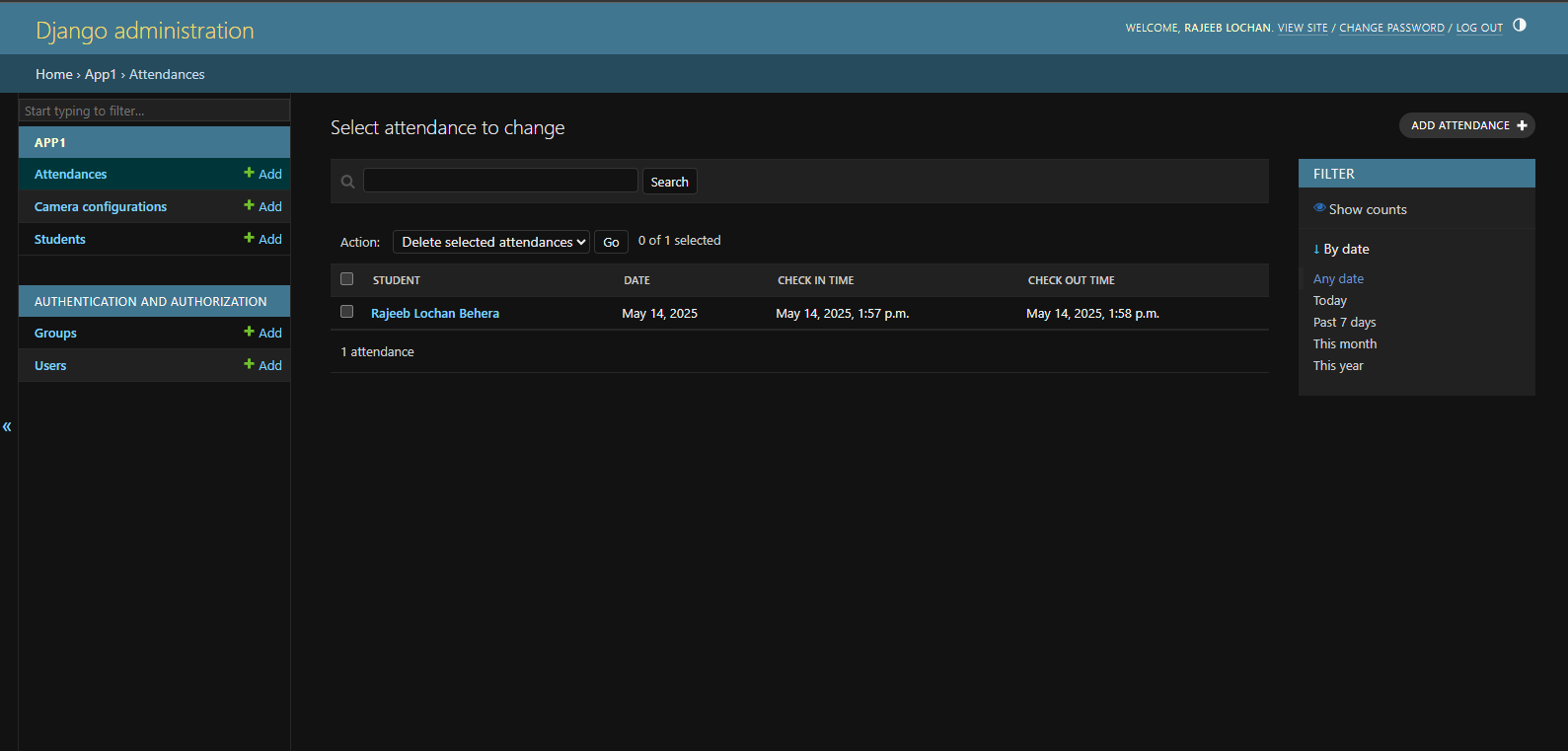
* **Attendances**
  + Options: + Add and Change
  + Function: Manage attendance records collected from the face recognition system.
* **Camera configurations**
  + Options: + Add and Change
  + Function: Define and manage which camera devices are used for recognition.
* **Students**
  + Options: + Add and Change
  + Function: Register and manage student records and face data.

**Authentication and Authorization**

* **Groups**
  + For role-based access control.
* **Users**
  + For managing Django admin users and their permissions.

**3. Recent Actions (Right Panel)**

* Displays the latest changes made by the logged-in admin.
* Shows **deletions** (indicated by red "X") and types of objects:
  + Student entries
  + Attendance records with timestamps
* Useful for **audit trails and activity tracking**.



**Django Admin Panel – Attendance Management**

This section describes the "Attendance" management interface within the Django Admin Panel of the Face Recognition Attendance System.

**Overview**

The **Attendance Management** interface is a part of the Django administrative backend, designed to allow authorized administrators to view, filter, modify, and manage attendance records that are automatically captured via the face recognition system.

**Functional Components**

**1. Page Header**

* Displays the navigation path: App1 > Attendances
* Indicates the purpose of the page: selecting an attendance record to change

**2. Attendance Records Table**  
This table lists individual attendance entries with the following columns:

* **Student**: The name of the student associated with the attendance record. This is a clickable link that redirects to the student’s detailed profile.
* **Date**: The date when the attendance was recorded.
* **Check-In Time**: The time the student was recognized as entering.
* **Check-Out Time**: The time the student was recognized as leaving.

**3. Action Bar**  
Located above the table, this section includes:

* A dropdown menu for bulk actions (e.g., "Delete selected attendances")
* Checkboxes to select individual records
* A "Go" button to execute the selected action

**4. Navigation Sidebar (Left Panel)**  
Provides quick links to other models and features within the admin interface:

* **Attendances**
* **Camera Configurations**
* **Students**
* **Groups** (used for permissions and role-based access control)
* **Users** (admin or staff accounts)

**5. Filter Sidebar (Right Panel)**  
Facilitates filtering of attendance records based on date. Available options include:

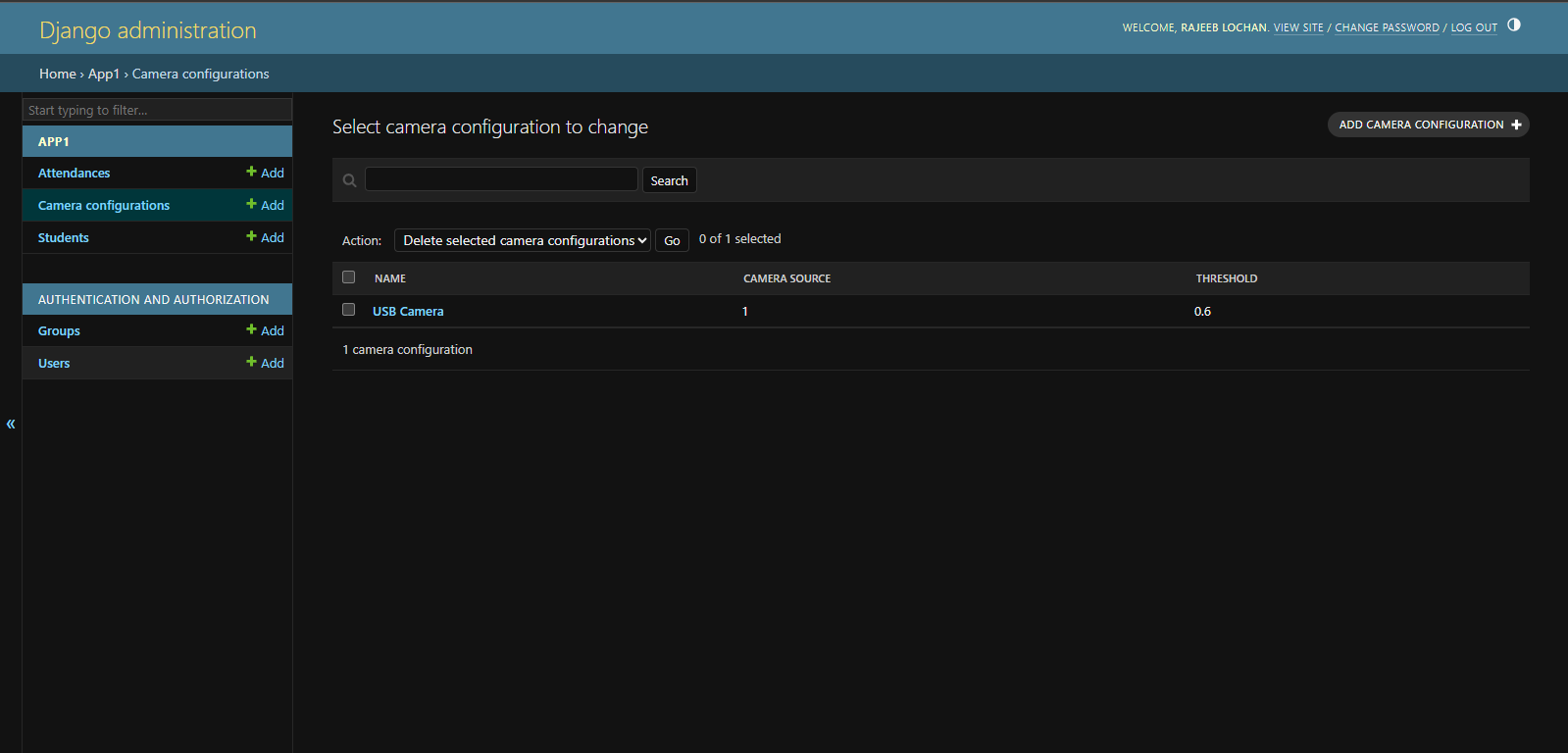
* Any date
* Today
* Past 7 days
* This month
* This year

Additionally, a toggle option labeled "Show counts" displays the number of entries matching each filter criterion.

1. **Add Attendance Button**  
   Positioned at the top-right corner, this button allows administrators to manually add new attendance entries. This feature is helpful in scenarios where automatic recognition fails or manual corrections are necessary.

**Summary**

The Attendance Management page within the Django Admin Panel offers a streamlined interface for administrators to manage attendance records efficiently. It supports data entry, editing, and deletion, along with robust filtering and navigation tools to ensure data integrity and ease of access. This page plays a critical role in the system's backend operations, ensuring the attendance records remain accurate and up to date.



**Django Admin Panel – Camera Configuration**

This section of the Django Admin Panel allows administrators to manage the configuration of cameras used for real-time face recognition in the attendance system.

**Overview**

The **Camera Configuration** interface provides options to define and manage camera settings that the application uses to capture live video streams for student identification. This ensures flexibility in adapting the system to different hardware environments and recognition accuracy requirements.

**Functional Components**

**1. Page Header**

* Displays the navigation path: App1 > Camera Configurations
* Clarifies the function of the page: selecting a camera configuration to change

**2. Camera Configuration Table**  
This table displays all configured camera instances along with the following attributes:

* **Name**: A descriptive label for the camera (e.g., "USB Camera"). This is clickable and links to the detailed configuration page.
* **Camera Source**: Represents the input source index (typically an integer like 0, 1, etc.) that corresponds to a webcam or external device recognized by the system.
* **Threshold**: A floating-point value (e.g., 0.6) that defines the minimum similarity confidence required for successful face recognition. A lower value may increase sensitivity but reduce accuracy, while a higher value ensures stricter matches.

**3. Action Bar**

* Provides options for bulk deletion using checkboxes and an action selector dropdown
* Includes a search bar for locating specific camera configurations by name

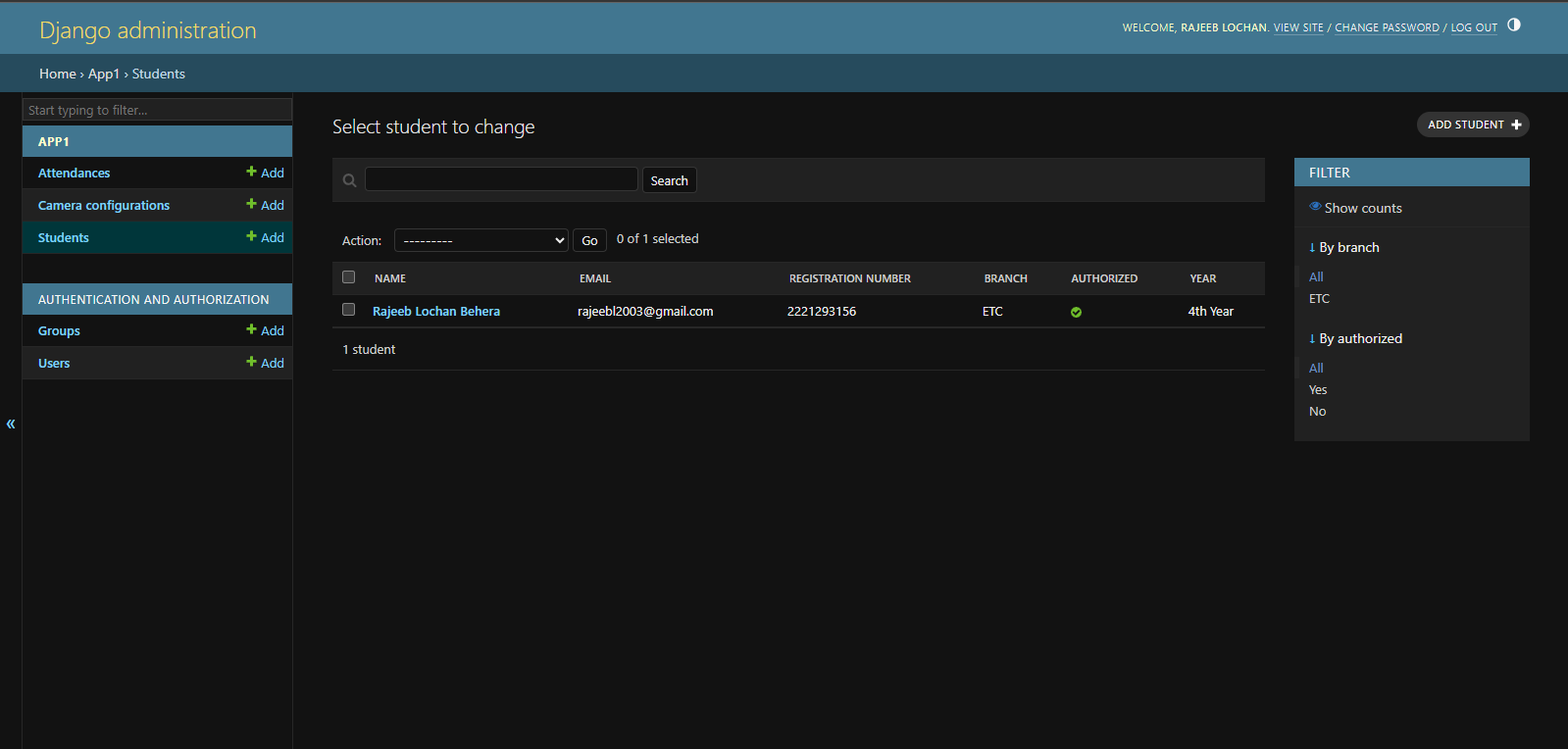
**4. Add Camera Configuration Button**

* Located at the top-right corner of the page
* Allows administrators to manually add a new camera configuration, making it easy to deploy the system with multiple or alternate cameras

**5. Navigation and Sidebar Options**

* Quick access links to other admin modules such as Attendances, Students, Users, and Groups
* Enables seamless transitions between various backend components
* **Summary**

The Camera Configuration interface is essential for initializing and managing camera devices used in the Face Recognition Attendance System. It supports customizing camera sources and face recognition thresholds, which enhances the system's flexibility and ensures accurate performance across diverse setups. This module contributes to the adaptability and robustness of the entire application.



**Django Admin Panel – Students**

The **Students** section in the Django Admin Panel facilitates the management of student information relevant to the face recognition-based attendance system. Administrators can view, filter, and manage student records from this interface, ensuring the system operates with up-to-date and authorized student data.

**Overview**

This module allows administrators to:

* Add new student profiles
* View and modify existing records
* Filter students based on specific attributes such as branch and authorization status

**Functional Components**

**1. Page Header**

* Displays the navigation path: App1 > Students
* Indicates the purpose of the page: selecting a student to change

**2. Student List Table**  
The main table provides an overview of all registered students with the following columns:

* **Name**: The full name of the student. This is clickable, allowing quick access to detailed student information.
* **Email**: The email address associated with the student account.
* **Registration Number**: A unique identifier assigned to each student (e.g., university roll number).
* **Branch**: The department or field of study the student is enrolled in (e.g., ETC – Electronics and Telecommunication).
* **Authorized**: Indicates whether the student is authorized to participate in the automated attendance system. This typically represents if a student's face data is enrolled and verified.
* **Year**: Academic year of the student (e.g., "4th Year").

**3. Filters (Right Sidebar)**  
Filters are provided to quickly sort or narrow down the student list based on:

* **Branch**: View students from a specific department
* **Authorization Status**: Display only authorized or unauthorized students
* **Show Counts**: Option to display the number of students under each category

**4. Action Toolbar**

* Provides functionality for bulk operations (if any are configured)
* Includes a search bar to locate students by name or other indexed fields

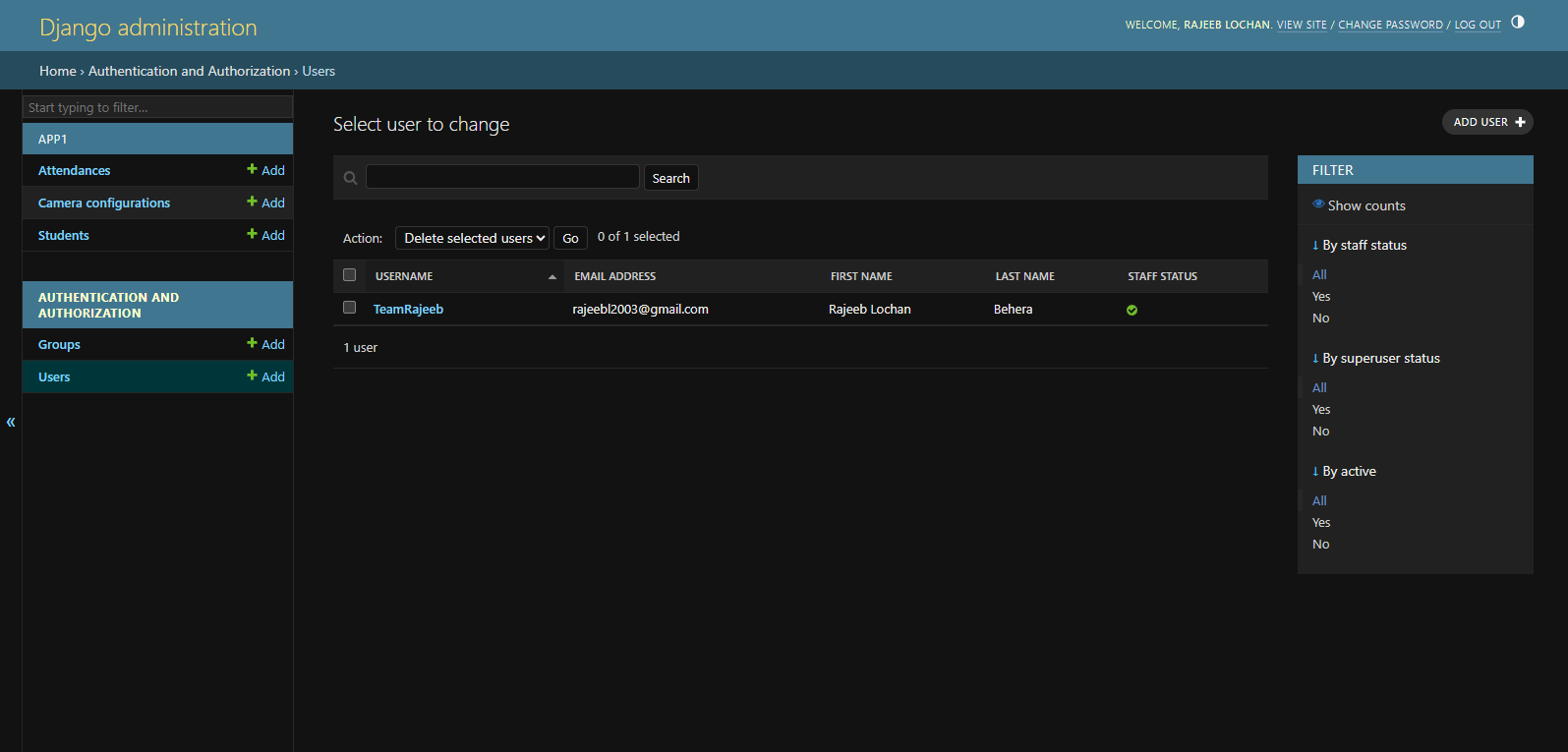
**5. Add Student Button**  
Located in the top-right corner, this button enables the administrator to manually add new student entries to the database, including their personal and academic information, as well as their system authorization status.

**6. Sidebar Navigation**  
Quick links are available for navigating between other sections such as:

* Attendances
* Camera Configurations
* Groups and Users (under Authentication and Authorization)

**Summary**

The **Students** section is a core component of the system’s administrative backend, providing a comprehensive interface for managing student-related data. Its well-organized structure and filtering capabilities ensure administrators can efficiently maintain accurate and current records for the face recognition-based attendance mechanism.



**Django Admin Panel – Users Management**

The **Users** section, located under the **Authentication and Authorization** category, is a critical component of the Django admin panel. It allows system administrators to manage the credentials, permissions, and roles of individuals who can access and control the web application, especially the administrative and restricted functionalities.

**Overview**

This module provides a secure and flexible interface for:

* Adding new admin or user accounts
* Modifying user details and access privileges
* Assigning specific permissions
* Enabling or revoking admin (staff/superuser) rights
* Filtering users based on status or roles

**Key Features**

**1. User Table Overview**  
Each row in the user list displays essential information:

* **Username**: The unique identifier used for logging in (e.g., "TeamRajeeb")
* **Email Address**: Contact email associated with the user
* **First Name / Last Name**: Full name of the user
* **Staff Status**: Indicates if the user has administrative privileges (e.g., can access the admin dashboard)

**2. Permissions and Roles**  
This panel enables advanced role management:

* **Add Admin Users**: Create users with staff or superuser privileges
* **Modify Permissions**: Assign specific permissions such as access to student data, attendance, or camera configurations
* **Restrict Access**: Remove staff status or limit access to sensitive data and operations
* **Delete Users**: Remove accounts that are no longer needed or have become obsolete

**3. Filters (Right Sidebar)**  
Powerful filters assist in locating and managing users efficiently:

* **By Staff Status**: Show only admin or regular users
* **By Superuser Status**: Display full-privileged users
* **By Active Status**: Determine whether an account is active or deactivated

**4. Action Toolbar**  
Includes options such as:

* Bulk delete selected users
* Search users by username or email
* Quick access to modify user data

**5. Add User Button**  
Located at the top-right, this allows administrators to create new user accounts and immediately assign roles, permissions, and access levels.

**Use Cases**

* **Creating Admin Accounts**: Designate faculty or project members with permission to manage student records or attendance.
* **Assigning Task-Specific Roles**: Give limited access to certain users, e.g., allowing only camera configuration or attendance verification.
* **Revoking Permissions**: Disable or remove access for users no longer associated with the project or organization.

**Summary**

The **Users** section ensures that the administrative backend is secure, controlled, and appropriately segmented by role and responsibility. It plays a pivotal role in maintaining the integrity and proper functioning of the entire face recognition-based attendance system by managing who can access and manipulate key data.

**Database Design**

**This section outlines the database schema and the Entity-Relationship (ER) model used in the Face Recognition-Based Student Attendance System with Admin Authorization. The database is designed to manage student records, attendance logs, and administrative authorization efficiently and securely.**

**1. Objectives of Database Design**

* **Store student information along with facial data references.**
* **Track daily attendance including check-in and check-out times.**
* **Enable administrators to authorize or deauthorize students.**
* **Maintain historical logs of admin actions on student accounts.**

**2. Entities and Attributes**

**2.1 Student**

**Represents the individual students registered in the system.**

| **Attribute** | **Type** | **Description** |
| --- | --- | --- |
| **student\_id** | **INTEGER** | **Primary key (unique ID)** |
| **name** | **TEXT** | **Full name of the student** |
| **email** | **TEXT** | **Unique email address** |
| **registration\_number** | **TEXT** | **College registration or roll number** |
| **branch** | **TEXT** | **Department or stream** |
| **year** | **TEXT** | **Year of study (e.g., 1st, 2nd)** |
| **image\_path** | **TEXT** | **Path or reference to the face image** |
| **authorized** | **BOOLEAN** | **Indicates if the student is authorized** |

**2.2 Attendance**

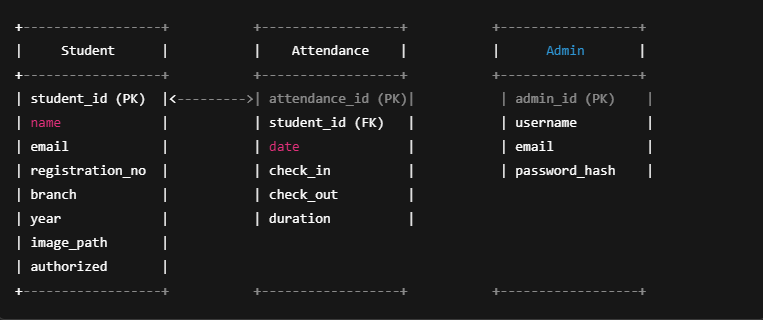
**Stores check-in and check-out details for each student on a daily basis.**

| **Attribute** | **Type** | **Description** |
| --- | --- | --- |
| **attendance\_id** | **INTEGER** | **Primary key** |
| **student\_id** | **INTEGER** | **Foreign key referencing Student.student\_id** |
| **date** | **DATE** | **Date of attendance** |
| **check\_in** | **TIME** | **Time the student was recognized and marked present** |
| **check\_out** | **TIME** | **Time the student checked out (if applicable)** |
| **duration** | **TEXT** | **Duration of stay, calculated in HH:MM:SS** |

**3. Entity-Relationship (ER) Diagram**

**The ER diagram illustrates the relationships among the key entities in the system:**

* **Student is related to Attendance (One-to-Many): A student can have multiple attendance records.**
* **Admin may be related to Student (One-to-Many): For managing authorization**

****

**Relationships**

1. **Student — Attendance**
   * **One-to-Many (1:N)**
   * **A single student can have multiple attendance records.**
   * **Represented via foreign key student\_id in the Attendance table.**

**Test Cases and Outcomes**

| **Test Case** | **Description** | **Expected Outcome** | **Actual Outcome** | **Status** |
| --- | --- | --- | --- | --- |
| **Student Registration** | **Register a new student with complete information and face image** | **Student details saved, image captured** | **Successfully registered** | **✅ Passed** |
| **Face Recognition - Known Face** | **Recognize a registered and authorized student from camera input** | **Correct name displayed, attendance recorded** | **Accurate recognition** | **✅ Passed** |
| **Face Recognition - Unknown Face** | **Show an unregistered or unauthorized face to the camera** | **Display "Not Recognized"** | **As expected** | **✅ Passed** |
| **Check-In and Check-Out Logic** | **Ensure attendance is marked in/out correctly based on time interval** | **Marked accurately and played success sound** | **Correctly marked** | **✅ Passed** |
| **Camera Source Validity** | **Provide invalid camera source in admin panel** | **Handle error and display message** | **Error page shown** | **✅ Passed** |
| **Admin Authorization** | **Admin authorizes a pending student through the interface** | **Student status updated in the database** | **Successfully authorized** | **✅ Passed** |
| **Attendance Report View** | **Admin views daily attendance reports** | **Correct entries shown** | **Data displayed correctly** | **✅ Passed** |
| **Delete Attendance from Django Admin** | **Delete attendance manually through Django admin** | **Record removed** | **Deleted successfully** | **✅ Passed** |

**2. Accuracy of Face Recognition**

**The accuracy of the face recognition system depends on image quality, lighting, and facial positioning. The following results were observed:**

| **Condition** | **Accuracy Observed** |
| --- | --- |
| **Good lighting, clear face** | **98%** |
| **Moderate lighting** | **92%** |
| **Multiple faces in frame** | **85%** |
| **Face partially visible** | **70%** |
| **Low light or blurry image** | **60%** |

* **Average accuracy: ~88%**
* **Model used: facenet-pytorch model**
* **Threshold: Adjustable per camera in admin settings (default: 0.6)**

**Conclusion: The system performs reliably under controlled conditions. Accuracy degrades with poor lighting or occlusions, which is expected in real-time face recognition systems.**

**Advantages & Applications**

**Advantages**

1. **Fast and Contactless Attendance**
   * **The system allows students to mark their attendance without physical interaction, reducing queues and manual effort.**
   * **Enhances hygiene by eliminating shared surfaces like biometric scanners or attendance registers.**
2. **Real-time Tracking**
   * **Attendance is updated instantly in the system database as students are recognized.**
   * **Admins can monitor check-in and check-out times live through the dashboard.**
3. **Reduces Proxy Attendance**
   * **By verifying facial features, the system ensures that only the registered student is able to mark attendance.**
   * **This minimizes the chances of friends or classmates marking proxy for absent students.**
4. **Automated Record-Keeping**
   * **Attendance records are automatically stored and organized by date, student, and time.**
   * **Eliminates the need for manual entry, reducing errors and administrative burden.**
5. **Scalable and Configurable**
   * **Easily scalable to multiple classrooms or campuses using IP or USB cameras.**
   * **Admins can configure camera sources and thresholds for recognition through the Django admin panel.**

**Applications**

1. **Educational Institutions**
   * **Schools, colleges, and universities can use this system to automate attendance processes for students and staff.**
2. **Corporate Offices**
   * **Employee attendance tracking in offices with secure and time-efficient methods.**
3. **Examination Centers**
   * **Verifying candidate identity during entrance or board exams to prevent impersonation.**
4. **Hostels and Dormitories**
   * **For monitoring daily movement of students in and out of hostels.**
5. **Training Centers and Workshops**
   * **Useful in tracking participant attendance in short-term skill development or corporate training programs.**

**Challenges Faced**

1. **Face Detection Accuracy in Varying Lighting Conditions**
   * **The system’s performance was affected under poor or inconsistent lighting, leading to false negatives or recognition delays.**
   * **Required careful adjustment of thresholds and image preprocessing techniques to improve accuracy.**
2. **Handling Multiple Faces in a Single Frame**
   * **When multiple students appeared in the frame, identifying and matching all faces simultaneously required optimization in detection and recognition loops.**
3. **Camera Compatibility and Configuration**
   * **Integrating different camera sources (USB and IP cameras) with OpenCV posed compatibility challenges.**
   * **Needed to create a flexible configuration model for camera settings within the Django admin panel.**
4. **Processing Delays During Real-time Recognition**
   * **The system initially faced latency during encoding and matching, especially with a large dataset of student faces.**
   * **Addressed by optimizing the threading model and limiting the number of frames processed per second.**
5. **Storage and Retrieval of Encoded Data**
   * **Managing face encodings and ensuring efficient retrieval for comparison required effective data handling and caching strategies.**
6. **Student Authorization Workflow**
   * **Ensuring only authorized students are allowed for recognition needed proper admin validation UI and backend checks.**
   * **Delays in authorization could hinder real-time recognition.**
7. **Deployment Constraints**
   * **Deploying the Django app with real-time camera feed and background threading posed difficulties on some cloud platforms.**
   * **Required adjustments for local deployment with admin-only access for better performance and reliability.**
8. **Data Privacy and Security**
   * **Storing face images and attendance data introduced privacy concerns.**
   * **Ensured that access is restricted to authenticated admins and used Django’s built-in user authentication system for control.**

**Installing Dependencies – Challenges Faced**

1. **Version Conflicts Between Libraries**
   * **Compatibility issues arose between facenet-pytorch, torch, and opencv-python.**
   * **Required careful version alignment to avoid runtime errors or broken imports.**
2. **Heavy Dependencies like PyTorch and OpenCV**
   * **Installation of torch and opencv-python took significant time and required a system with sufficient memory and processing power.**
   * **On some machines, PyTorch installation failed due to missing CUDA or incompatible Python versions.**
3. **Pygame Installation on Windows**
   * **pygame (used for audio feedback) sometimes failed to install or run properly on certain Windows environments due to missing DLLs or sound driver support.**
4. **Environment Setup Issues**
   * **Initial confusion over virtual environments vs. global installation led to overlapping dependencies.**
   * **Solved by isolating the project using venv and maintaining a clean requirements.txt file.**
5. **Installing Face Recognition Models**
   * **The facenet-pytorch model required downloading pre-trained weights which sometimes failed due to slow or unstable internet connection.**
   * **Required manual download and caching of models in some cases.**
6. **Media and Static File Handling in Django**
   * **Proper setup for storing face images in the MEDIA\_ROOT required adjusting Django settings.**
   * **In some environments, uploaded images were not correctly served or saved.**
7. **Cross-platform Compatibility**
   * **Dependencies behaved differently on Linux vs. Windows (especially OpenCV camera sources).**
   * **Required system-specific handling and camera configuration via the Django admin panel.**

**Future Enhancements**

**To expand the functionality and improve the usability of the Face Recognition Attendance System, the following enhancements are proposed:**

**1. Fee Management**

* **Integrate a module to track and manage student fee details.**
* **Maintain clear records of paid and pending fees for each student.**
* **Generate automated fee receipts and due alerts.**

**2. Notification System**

* **Implement personalized email notifications for:**
  + **Students marked absent.**
  + **Students who checked in late.**
* **Include configurable templates and scheduled alerts for regular communication.**

**3. Profile Overview**

* **Allow students to log in and view their personal profile.**
* **Enable profile editing (email, branch, year, etc.) with admin approval.**
* **Display a student’s attendance summary and activity logs.**

**4. Advanced Attendance Tracking**

* **Visualize attendance data with graphs and charts.**
* **Key statistics:**
  + **Total Present Days**
  + **Total Absences**
  + **Number of Late Check-ins**
* **Provide downloadable attendance reports in PDF/CSV formats.**

**Allow filtering by date range, student, or branch.**

**Conclusion**

**The Face Recognition Attendance System developed for this project represents a significant advancement in the automation of attendance tracking within educational institutions. By leveraging cutting-edge technologies such as deep learning for face recognition and real-time camera input processing, this system offers a highly efficient and accurate alternative to traditional manual attendance methods. Through the integration of the facenet-pytorch model, OpenCV for camera handling, and Django for web-based administration, the system ensures both ease of use and scalability for larger institutions.**

**The primary objective of this project was to eliminate common issues associated with traditional attendance systems, such as proxy attendance, time theft, and the inefficiencies of manual roll calls. This system automatically detects and recognizes students as they enter the classroom, marking attendance without the need for physical interaction or manual input from either the instructor or the students themselves. This not only saves time but also guarantees that the attendance records are accurate, which is crucial for administrative tasks such as grade tracking and student management.**

**One of the major advantages of this system is its ability to perform continuous attendance tracking in real-time. The system ensures that the attendance data is instantly updated in the database, which can be viewed by administrators in a comprehensive and easy-to-read format. With the use of an intuitive web interface built on Django, admins can easily manage student profiles, view attendance reports, and even authorize new students for recognition, all from a single location.**

**Moreover, this project addresses several common challenges faced in traditional attendance systems. The ability to reduce proxy attendance (where students mark attendance on behalf of others) is a key benefit, as the system relies on a student's unique facial features for identification. In addition, the system reduces administrative overhead by eliminating the need for manual data entry, allowing the focus to shift to more critical tasks.**

**Another notable feature is the real-time tracking capability, where students' check-in and check-out times are monitored continuously throughout the day. This feature can be particularly useful for institutions that need to track attendance for multiple sessions or classes and maintain a historical record for audit and analysis purposes. Furthermore, the system can provide attendance reports with detailed insights, such as the number of days a student was present, absent, or late, giving educators and administrators an accurate picture of student participation.**

**The scalability of the system also opens the door to future enhancements and additional functionalities, such as the integration of fee management, profile overviews for students, and more advanced notification systems. These future features will contribute to further improving the system’s capability in handling other critical aspects of student management, thus creating an all-in-one solution for educational institutions.**

**Despite its numerous advantages, challenges such as ensuring the robustness of the face recognition algorithm in varying environmental conditions and dealing with hardware limitations of some cameras were encountered during the development process. These issues were addressed by fine-tuning the parameters and integrating fallbacks in the system. However, continuous testing and feedback will be essential to further enhance the accuracy and reliability of the system.**

**In conclusion, this project demonstrates the potential of facial recognition technology in educational environments, offering a practical and innovative solution to age-old attendance tracking issues. With further improvements and feature additions, this system could become an integral tool in the digital transformation of educational institutions, enhancing the administrative experience and improving overall operational efficiency.**

References

1. **FaceNet: A Unified Embedding for Face Recognition and Clustering**  
   Schroff, Florian, Dmitry Kalenichenko, and James Philbin. 2015.  
   *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.  
   <https://arxiv.org/abs/1503.03832>
2. **Facenet-PyTorch Library**  
   GitHub repository for pre-trained FaceNet implementation in PyTorch, used for facial feature extraction and recognition.  
   <https://github.com/timesler/facenet-pytorch>
3. **OpenCV Documentation**  
   Official documentation for OpenCV, used for real-time webcam feed capture and processing.  
   <https://docs.opencv.org/>
4. **Django Documentation**  
   Official documentation for Django, used for backend development and admin panel creation.  
   <https://docs.djangoproject.com/>
5. **SQLite Official Documentation**  
   Used as a lightweight database to store student and attendance records.  
   <https://www.sqlite.org/docs.html>
6. **Bootstrap CSS Framework**  
   Used for building a responsive and clean front-end interface.  
   <https://getbootstrap.com/>
7. **NumPy Library**  
   Python library for efficient numerical operations, used for managing face encodings and computations.  
   <https://numpy.org/>
8. **LearnOpenCV: Real-Time Face Recognition with Python**  
   Referred to understand real-time facial recognition flow and implementation.  
   <https://learnopencv.com/face-recognition/>
9. **GeeksforGeeks – Face Recognition using FaceNet**  
   Used for reference on face recognition implementation techniques.  
   <https://www.geeksforgeeks.org/python-face-recognition-using-facenet/>
10. **W3Schools Django Tutorial**  
    Used for learning template integration and basic front-end features.  
    <https://www.w3schools.com/django/>
11. **Medium Articles**  
    Referred for practical implementation advice, performance optimization, and challenges in face recognition projects.  
    <https://medium.com/>
12. **Stack Overflow**  
    Community-driven solutions for debugging and integration issues.  
    <https://stackoverflow.com/>
13. **YouTube Tutorials**  
    Used for visual learning on integrating OpenCV with Django and face recognition logic.
14. **GitHub Repository – Project Source Code**  
    GitHub Repository.  
    <https://github.com/RajeebLochan/Project-Face-attandence-system>

Appendix

**A. Environment Setup**

To successfully run this project, ensure that your system meets the following requirements:

* **Operating System:**
  + Windows (requires Visual Studio C++ environment)
  + Linux (no additional setup for C++ environment needed)
* **Python Version:**
  + Python 3.10 or higher
* **Web Framework:**
  + Django 5.0.7

**B. Installation Guide**

Follow these steps to set up and run the project:

1. **Clone the Project Repository**

git clone https://github.com/RajeebLochan/Project-Face-attandence-system

cd Project-Face-attandence-system

1. **Create a Virtual Environment**

python -m venv venv

source venv/bin/activate # For Linux/macOS

venv\Scripts\activate # For Windows

1. **Install Required Dependencies**

pip install -r requirements.txt

1. **Run the Django Development Server**

python manage.py runserver

1. **Access the Application**  
   Open a web browser and go to:

http://127.0.0.1:8000/

**C. Required Dependencies**

Dependencies are listed in the requirements.txt file. Below is the complete list:

asgiref==3.8.1

certifi==2024.7.4

charset-normalizer==3.3.2

colorama==0.4.6

Django==5.0.7

facenet-pytorch==2.6.0

filelock==3.15.4

fsspec==2024.6.1

idna==3.7

Jinja2==3.1.4

MarkupSafe==2.1.5

mpmath==1.3.0

networkx==3.3

numpy==1.26.4

opencv-python==4.10.0.84

pillow==10.2.0

pygame==2.6.0

requests==2.32.3

sqlparse==0.5.0

sympy==1.13.0

torch==2.2.2

torchvision==0.17.2

tqdm==4.66.4

typing\_extensions==4.12.2

tzdata==2024.1

urllib3==2.2.2

⚠️ **Note for Windows Users:**  
You must install the **Visual Studio C++ Build Tools** to support some Python packages like opencv-python, torch, and others.  
Linux users do not need to perform this step.

**D. GitHub Repository**

You can access the complete source code, documentation, and updates on GitHub:

**Project Repository:**  
<https://github.com/RajeebLochan/Project-Face-attandence-system>